UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERÍA

OPTIMIZACIÓN DE RUTAS DE VEHÍCULOS BASADO EN UN ALGORÍTMO HÍBRIDO CON SIMULACIÓN DE BLOQUEOS EN CONDICIONES DE TRÁFICO

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA

INGENIERÍA DE SISTEMAS – INGENIERÍA INDUSTRIAL

P R E S E N T A :

LUIS FERNANDO MARTÍNEZ MOSQUEIRA

TUTOR:
DR. JUAN MANUEL ESTRADA MEDINA

MÉXICO, D.F. SEPTIEMBRE 2009

JURADO ASIGNADO:

Presidente: Dra. Idalia Flores de la Mota

Secretario: Dr. Ricardo Aceves García

Vocal: Dr. Juan Manuel Estrada Medina

1er. Suplente: Dra. Mayra Elizondo Cortés

2do. Suplente: Dra. Cozumel Allanec Monroy León

Lugar donde se realizó la tesis:

Facultad de Ingeniería, Ciudad Universitaria, D.F.

TUTOR DE TESIS:
DR. JUAN MANUEL ESTRADA MEDINA

_____

FIRMA

DEDICATORIA:

A mi madre, Blanca Mosqueira Paredes por ser el motivo y la inspiración para seguir adelante y por apoyarme siempre para realizar mis sueños.

A mi hermano Jorge, por su amistad, valiosos consejos y por estar siempre ahí en los momentos buenos y malos.

A mis tíos Luis, Fernando, Jorge, Yolanda, Isabel, Rocío Mosqueira por su gran paciencia y gran cariño.

A todos mis profesores de Ingeniería Mecatrónica e Industrial.

A toda mi familia.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mi tutor Dr. Juan Manuel Estrada Medina por su gran paciencia, no podría haberlo hecho sin su guía, sin sus valiosas enseñanzas y sabios consejos. Ha sido maravilloso trabajar con usted.

Agradezco a todos mis sinodales, quienes pusieron todo su empeño en la revisión de esta tesis; su buen trabajo ha hecho el mío sencillo.

Al CONACYT por haber apoyado económicamente mis estudios de maestría y por alimentar mi espíritu de investigador.

Doy las gracias a mis viejos amigos que contribuyeron con sus espléndidos consejos: Choch, Euch, Tenoch, Alice, Gina, Kimik, Negrit, Fer, Gabo, George, Daniel, Gustavo y Manson.

A mi hermano y gran amigo, el dios de la cirugía plástica Jorge M. Mosqueira, que de verdad leyó mi tesis y me hizo creativas sugerencias. Y no sólo eso, siempre estuvo allí cuando lo necesitaba, eres un gran hermano!

A Koz por el profundo lazo cósmico que nos une, gracias por inspirarme y estar siempre presente!

Y finalmente, a mi banda de rock LUPANAR por darme mi dosis musical necesaria para existir y por confiar en mi todos estos años.

ÍNDICE

Introducción

La necesidad de transportar determinados productos o servicios de un lugar a otro demanda que las empresas requieran de un importante proceso logístico en cuanto a la programación y designación de rutas para sus vehículos destinados a distribuir, repartir o proporcionar algún servicio de manera eficiente. Sin embargo, existen factores o situaciones imprevistas (rutas bloqueadas por alguna manifestación, accidentes, etc.) que impactan el costo, por ejemplo, el consumo excesivo de combustible y demoras en los tiempos de entrega. En este contexto, es de suma importancia poner atención a este tipo de problemas debido a que se pueden utilizar herramientas computacionales las cuales son de utilidad para modelar estos eventos imprevisibles. En este trabajo se utilizó el ambiente **MATLAB** para desarrollar un algoritmo y simular bloqueos estáticos y dinámicos en rutas de vehículos, ya que permite manejar con facilidad matrices de gran dimensión [1][16][21].

Problemática

En las grandes urbes, el *tráfico vehicular* está presente en casi todas las actividades cotidianas de la gente, y ocasiona numerosos fenómenos entre los que destacan especialmente los congestionamientos. Estos últimos, algunas veces tienen causas obvias, por ejemplo, una manifestación obstruyendo la circulación, desvíos por obras de mantenimiento, o un accidente. Estas son algunas de las razones más importantes por las cuales se complica el tránsito vehicular. Sin embargo, también influye en gran medida el comportamiento en los conductores de los vehículos. En efecto, se ha observado que en condiciones de tráfico denso, un conductor tiende a mantener una distancia que le parece "*cómoda"* y segura entre su vehículo y el que tiene inmediatamente frente de él. Si fuera el caso, de que este último llegara a frenar un poco, dicho conductor haría lo mismo, pero en el lapso que tarda en reaccionar para frenar, la distancia *"cómoda"* de separación disminuye y por lo tanto, decrece aún más su velocidad. Esto a su vez produce una reacción en cadena con los conductores que le anteceden; hasta que el tráfico se detiene formando un embotellamiento. Como se puede advertir, son muchos los factores involucrados en una congestión de tráfico, sin mencionar la densidad del flujo vehicular, los límites de velocidad, semáforos, etc. Una teoría que trate de modelar este sistema dinámico es, por supuesto, necesariamente compleja.

Aunado a lo anterior, las obstrucciones repentinas que se presentan a través de un flujo vehicular, generan múltiples efectos negativos sobre los procesos referentes al transporte o distribución de productos o servicios, por ejemplo, los retrasos en los tiempos de entrega, el incremento en los gastos de combustible, pérdidas en los costos de oportunidad, desgaste de los vehículos (mayor frecuencia de reparaciones), etc. Además de resultar una actividad frustrante y estresante para los automovilistas debido a que se está inactivo por mucho tiempo en un mismo lugar. Estas incertidumbres dificultan a las empresas la tarea de designar las mejores rutas para sus vehículos encargados de transportar sus productos.

El problema de ruteo de vehículos ha sido investigado desde un enfoque estático, y puede formularse como una instancia del Problema del Agente Viajero (**TSP,** por sus siglas en inglés *Traveling Salesman Problem*), el cual se enuncia de la siguiente manera. Un agente viajero, partiendo de su ciudad de origen, debe visitar exactamente una vez cada ciudad de un territorio y regresar al punto de partida de tal manera que la distancia total recorrida sea mínima. El modelo **TSP** tiene muchas aplicaciones en problemas reales, por ejemplo, en la logística de distribución de productos se utiliza para determinar las rutas que deben de seguir los vehículos para realizar su entrega en el menor tiempo posible. Sin embargo, en estos estudios no se atienden los bloqueos súbitos, ni las congestiones de tráfico en las rutas.

En este contexto, la aportación del presente trabajo con respecto al problema antes mencionado, fue que se analizó el asunto del ruteo de vehículos desde un punto de vista *dinámico* lo que implicó un aumento considerable en su complejidad. Por ejemplo, encontrar una solución óptima supone no sólo encontrar un recorrido factible de los clientes, sino también la selección dinámica de la ruta óptima dentro de un amplio rango de posibles rutas en un instante dado. En esta investigación se simularon los bloqueos en las rutas tomando como marco el modelo del conductor inteligente (**IDM**, por sus siglas en inglés *Intelligent Driver Model*) ya que es el que más se adecua a la realidad. Para ello se construyó un algoritmo híbrido cuyo objetivo fue encontrar rutas alternas en caso de presentarse bloqueos inesperados en alguna de las rutas, esto con el fin de dar soporte a las empresas en el proceso de toma de decisiones, lo cual puede tener un fuerte impacto económico.

Objetivo

Diseñar un algoritmo híbrido que encuentre la ruta óptima de un flujo vehicular simulado donde repentinamente ocurre un bloqueo de cierto tipo a lo largo del recorrido.

El objetivo se alcanzó a través del diseño de un algoritmo híbrido de 4 etapas, el cual se ilustra en la siguiente figura:

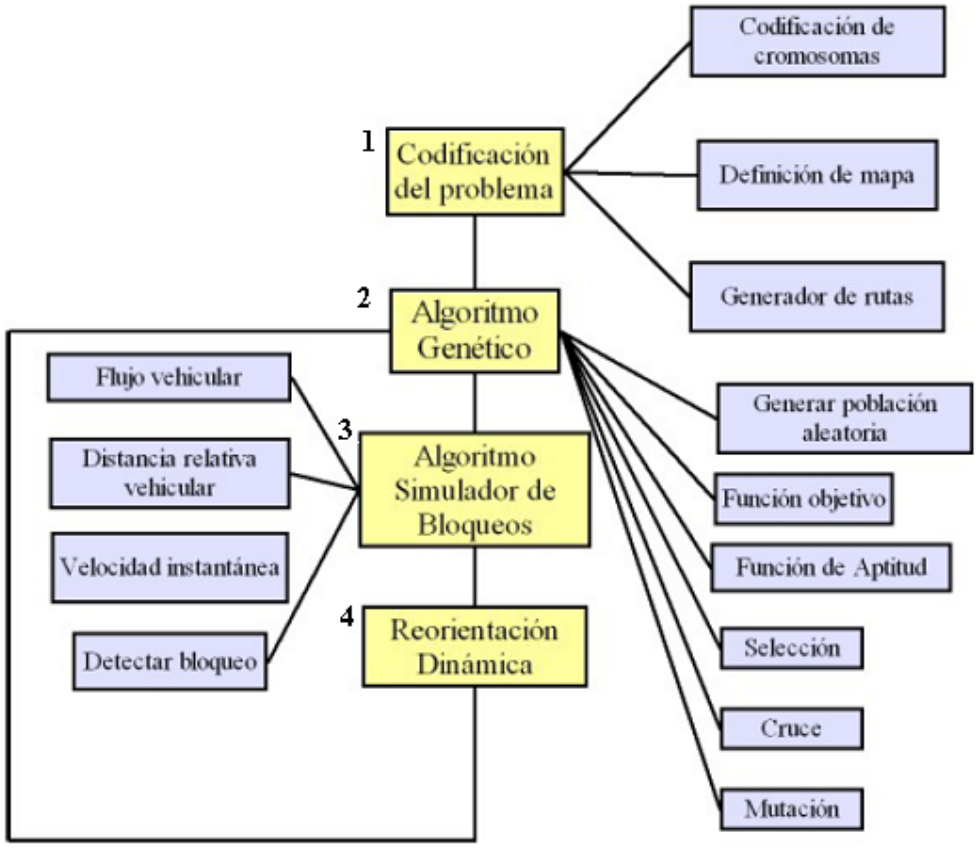


Fig. 1 Esquema general del algoritmo híbrido propuesto.

1.- Codificación: En esta etapa se planteó el problema de forma codificada, de manera que se pudiera manipular computacionalmente bajo un ambiente en **MATLAB**. Dentro de esta etapa también se implementaron los siguientes elementos:

- *Definición de mapa:* Se definió un mapa digital imaginario en el cual se especificaron los parámetros físicos de las calles (longitud, sentido, posición, etc.).
- *Generador de rutas:* Se creó una matriz con todas las rutas posibles que conectan con un determinado destino (espacio de búsqueda).

2.- Algoritmo genético: Permitió explorar e intensificar ciertas regiones del espacio de búsqueda para aproximar la mejor solución mediante la construcción de una función objetivo. También evitó que el algoritmo se estancara en un mínimo local.

3.- Simulador de bloqueos dinámico: Pone a prueba el algoritmo bajo distintos escenarios de flujo vehicular. La simulación se realizó con base en el "*modelo del conductor inteligente"* **IDM**, este modelo intenta describir el comportamiento de un conductor al volante de un vehículo en presencia de algún bloqueo, y toma como parámetros características propias de los conductores, por ejemplo, el tiempo de reacción de frenado, la distancia de separación "cómoda" que guarda con respecto a los demás automóviles, etc. El modelo se analiza con detalle en el capítulo III.

4.- Reorientación dinámica: La reorientación dinámica es la solución final obtenida por el algoritmo en un cierto instante. Esta solución representa el recorrido óptimo (si es que existe) libre de bloqueos el cual se evalúa constantemente para medir su desempeño.

- El presente trabajo consta de cuatro capítulos, en el capítulo 1 se plantea el problema de ruteo de vehículos con bloqueos en condiciones de tráfico y se explica el enfoque dinámico sobre el **TSP**. El capítulo 2 describe las características generales de los algoritmos genéticos, como son la codificación del problema, operadores genéticos, métodos de selección, esquemas de reemplazo, etc. El capítulo 3 expone con detalle el desarrollo del algoritmo propuesto incluyendo la penalización de la función objetivo y la implementación de los operadores genéticos especiales de reparación, mutación y cruce. El capítulo 4 detalla la metodología experimental y los resultados obtenidos. Finalmente, la sección de conclusiones, las referencias bibliográficas y los anexos.

Capítulo I

Problema de ruteo de vehículos con bloqueos en condiciones de tráfico

"Jamás se desvía uno tan lejos como cuando cree conocer el camino"
Proverbio Chino

Antecedentes

Este capítulo tiene un carácter introductorio al problema y pretende ofrecer al lector una mayor información sobre el tema que aborda el presente trabajo. Imagine que un vehículo repartidor de cierto producto necesita realizar una entrega a un cliente **C1** partiendo desde un almacén (**F**ig. 2). Suponga que el conductor del auto conoce 2 posibles rutas (**R1** y **R2**) para realizar su viaje, y que además conoce la longitud total de ambos caminos. Entonces, lo más probable es que decida escoger la ruta de menor longitud. Sin embargo, esto no siempre es lo más conveniente, pues podría darse el caso de que en determinado momento dicha ruta estuviera congestionada por el tráfico o quizás bloqueada por alguna manifestación o un accidente, lo cual retrasaría la entrega y por lo tanto generaría molestias al cliente, y éste quizás no vuelva a contratar el servicio.

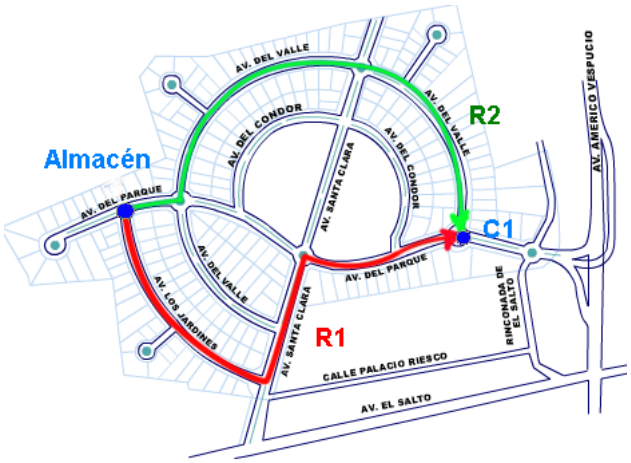


Fig. 2 Posibles rutas para realizar la entrega

Como se puede notar, la *longitud* de una ruta es información suficiente para elegir cuál es el camino más adecuado. En este trabajo definimos una *distancia dinámica* que considera los bloqueos súbitos y las condiciones del tráfico, las cuales se determinan por medio de un flujo vehicular simulado mediante el modelo del conductor inteligente **IDM**. De esta manera, la *distancia dinámica* depende del instante en el que se desee realizar el viaje, por ejemplo, en una ciudad a horas pico esta será muy grande, y por el contrario a altas horas de la noche puede ser muy baja. En otras palabras, la distancia dinámica penaliza las rutas que presentan un alto congestionamiento vehicular o bloqueos súbitos. Esta penalización se logra sumando una cantidad "*muy grande*" (valor de penalización) a la longitud de la ruta, y esta dependerá del número de bloqueos presentes en la ruta. Si no existiera ningún bloqueo, la distancia dinámica coincide con la longitud real de la ruta y por tanto deja de ser función del tiempo. El criterio utilizado para considerar los bloqueos en la ruta se explicará más adelante en la parte del modelado del problema. Estas ideas

expresadas en términos matemáticos, sería así: Denote una ruta cualquiera por la letra *k*, entonces la distancia dinámica estaría representada por la siguiente ecuación:

$$d_k(t) = L_k + P * Nb_k(t)$$

Donde:

$d_k(t)$ = Distancia dinámica de la ruta *k*

L_k = Longitud de la ruta *k*

P = Valor de penalización

$Nb_k(t)$ = Número de bloqueos en la ruta *k* en el instante *t*

Ahora suponga, que se requieren visitar *n* clientes sólo una vez y regresar al punto de partida. Entre cada cliente existen muchas rutas disponibles por las que se puede realizar el viaje, obviamente algunos clientes tendrán más rutas que otros o incluso el camino de ida puede ser distinto que el de regreso (rutas dirigidas). En este contexto, el problema consiste en encontrar el recorrido óptimo para visitar a todos los clientes seleccionando la ruta con menor *distancia dinámica*.

Enfoque dinámico sobre el problema del agente viajero TSP

Considerando lo anterior, el problema presenta una estructura similar a un **TSP**, en efecto, ambos problemas se modelan como una optimización combinatoria de recorridos para visitar todos los clientes de una ciudad sin repetir ninguno de ellos con la menor distancia posible, la diferencia estriba en que el problema de ruteo con bloqueos considera *distancias dinámicas* entre los clientes, mientras que en el **TSP** dicha distancia es *fija*. Por tal motivo tomaremos este último como modelo inicial y estudiaremos sus principales características y métodos de resolución. Luego introduciremos el concepto de *distancia dinámica* en dicho modelo.

- *Descripción del TSP y métodos de solución*

El **TSP** consiste en visitar *n* ciudades de un territorio exactamente una vez y regresar al punto de partida. El recorrido que el agente viajero realiza se le llama "*tour"* y la distancia entre cada ciudad viene dada por la matriz de distancias **D** de orden *n*x*n*, donde cada elemento d_{ij} de la matriz representa la distancia que hay entre las ciudades *i* y *j* [3]. También existe una variante asimétrica del problema, en donde la distancia de *i* a *j* puede ser distinta a la distancia de *j* a *i* ($d_{ij} = d_{ji}$ o $d_{ij} \neq d_{ji}$) [3] [11] [16]. El objetivo es minimizar la distancia recorrida por el agente viajero, que en términos matemáticos significa minimizar una función objetivo. En efecto, sea

$d(c_i, c_{i+1})$ la distancia de la ciudad c_i a la ciudad c_{i+1} , y n el número de ciudades a recorrer, entonces la función a minimizar es:

$$\min \left\{ \sum_{i=1}^{n-1} d(c_i, c_{i+1}) + d(c_n, c_1) \right\}$$

Note que el término $d(c_n, c_1)$ cierra el ciclo del *tour*, es decir, el agente viajero regresa al punto de partida.

El **TSP** fue formulado por primera vez de forma matemática en 1930 y desde entonces ha sido uno de los problemas más estudiados en el campo de la optimización, una de las subdisciplinas de la Investigación de Operaciones [2]. El **TSP** con ligeras modificaciones, aparece en muchas aplicaciones prácticas que van desde la logística, la manufactura de microchips hasta el secuenciamiento del DNA [5]. En estas aplicaciones, el concepto de *ciudad* representa, por ejemplo, clientes, puntos de soldadura, o genes, y el concepto de *distancia* puede representar tiempos de viaje, costos o medidas de similitud entre fragmentos de DNA. En estas aplicaciones generalmente se añaden restricciones, por ejemplo, recursos limitados o ventanas de tiempo lo que aumenta considerablemente su complejidad [16]. En este trabajo los conceptos de ciudad y distancia representan a los clientes y la distancia dinámica respectivamente.

El procedimiento más directo para resolver el **TSP**, consiste en evaluar todas las posibles combinaciones de recorridos y tomar como solución aquella que utiliza la menor distancia [2]. Sin embargo, esta técnica no es factible ya que el número de posibles combinaciones viene dado por el factorial del número de ciudades ($n!$), lo cual dificulta obtener una solución para valores de n grandes, o moderados, incluso con los medios computacionales actualmente a nuestro alcance. Por ejemplo, si un ordenador fuese capaz de calcular la longitud de cada combinación en un microsegundo, tardaría algo más de 3 segundos en resolver el problema para 10 ciudades, algo más de medio minuto en resolver el problema para 11 ciudades y... 77 años en resolver el problema para sólo 20 ciudades! [4]. Se ha demostrado desde el punto de vista de la complejidad computacional que el **TSP** es del tipo **NP-*Completo***, es decir, se requiere un tiempo elevadísimo para obtener la solución óptima incluso en problemas pequeños [1][3]. Debido a las dificultades que complican encontrar la solución exacta del **TSP**, en la práctica se emplean técnicas que proporcionan aproximaciones aceptables para resolver el problema [5].

Una técnica comúnmente utilizada para aproximar la solución del **TSP** consiste en realizar una búsqueda local en la vecindad de una solución inicial, si se encuentra una mejor solución en el entorno, ésta reemplaza la solución inicial y se continúa con el proceso hasta que ya no sea posible mejorar la solución actual. Esta estrategia permite aproximar soluciones muy rápidamente, sin embargo, puede quedar atrapada en óptimos locales y su solución final depende en gran medida de la solución inicial. Por tal motivo, en las últimas décadas se han desarrollado diferentes métodos alternativos para aproximar la solución óptima del **TSP**, es el caso de las técnicas *metaheurísticas*, las cuales ofrecen soluciones aproximadas a problemas generales del tipo **NP** *completos*, sin necesidad de recorrer o explorar todo el espacio de búsqueda [23]. Estas técnicas utilizan la experiencia ganada en búsquedas previas (memoria) para intensificar nuevas búsquedas en regiones prometedoras en dicho espacio, y su potencial radica en que incorporan mecanismos para escapar de óptimos locales [23] [24]. La mayoría de las técnicas metaheurísticas se inspiran en analogías de fenómenos físicos (templado simulado), de la etología (colonia de hormigas), y de la biología (algoritmos genéticos).

El presente trabajo se enfoca en los Algoritmos Genéticos (**AG**) ya que han demostrado tener gran potencial en aplicaciones exitosas, por ejemplo, en el campo de la planeación de rutas de vehículos [3] [18] [22], y a diferencia de otros métodos basados en gradientes, dichos algoritmos genéticos sólo requieren conocer los valores de la función objetivo, lo que permite tratar problemas con espacios de búsqueda complejos de gran dimensión y poco entendidos. Por otra parte, los **AGs** son relativamente fáciles de programar y muy flexibles para hibridar con otras estrategias de optimización que agregan información del espacio de búsqueda para aumentar su rendimiento y generalmente encuentran soluciones con un alto grado de aproximación. En efecto, con el ajuste adecuado de sus parámetros, los algoritmos genéticos pueden alcanzar un gran balance entre la exploración y la intensificación sobre el espacio de búsqueda. La exploración permite que el algoritmo recorra varias zonas del espacio de soluciones, con lo que se evitan los óptimos locales. Por otro lado, la intensificación ayuda a moverse en el espacio circundante de una posible solución para determinar la que más se acerque al óptimo en dicha vecindad.

Los **AGs** son técnicas de búsqueda basadas en la mecánica de la selección natural y la genética [3]. En este proceso generacional, una población inicial evoluciona a partir de ciertos puntos del espacio (*individuos*) hacia mejores regiones del espacio de búsqueda [5]. La evolución de la población se realiza mediante la aplicación de operadores genéticos probabilísticos de selección, cruce (*crossover*) y mutación. Su utilidad viene dada por la suposición de que diferentes partes de la solución óptima pueden ser descubiertas independientemente para luego ser combinadas y formar mejores soluciones. Los algoritmos genéticos se abordarán con más detalle en el Capítulo II.

Estado del arte

Durante la revisión de la literatura sobre la construcción de algoritmos de ruteo dinámico, se observó que este aspecto dinámico ha sido poco investigado, en contraste con los estudios de tipo estático [2] es decir, no se encontró actualmente un tratamiento del problema como el que se realizó en este trabajo. Sólo existen referencias acerca de problemas en el área de optimización y planeación de rutas [3], sin embargo, en ningún caso se aborda la cuestión de los bloqueos por situaciones de tráfico desde una perspectiva dinámica [5]. Las empresas finlandesa *Nokia* y la estadounidense *Google* empiezan a ofrecer en algunos países un servicio de información sobre las condiciones del tráfico en tiempo real [20], dichas firmas pretenden utilizar estos datos para establecer redireccionamientos dinámicos en las rutas de los vehículos y evitar posibles congestiones de tráfico [19]. A pesar de estos avances en el tema, no existe información suficiente sobre el funcionamiento de los algoritmos para resolver este tipo de problemas y por tanto se carece de criterios para hacer comparaciones con nuestro algoritmo [23].

Modelado del problema

A continuación se presentan algunas definiciones que serán necesarias para introducir el concepto de *distancia dinámica* y construir el modelo.

- Ruta

Se define una ruta como un conjunto ordenado de calles que conectan dos clientes específicos. En la **F**igura 3 se muestran 3 rutas disponibles (*A, B y C*) que conectan los clientes c_1 y c_2, como se puede notar cada ruta tiene asociado una secuencia de calles bien definidas, por ejemplo, para ir de c_1 a c_2, la ruta A queda completamente especificada por el conjunto $A_{c1,c2} = \{4,5,6\}$, del mismo modo la ruta B por $B_{c1,c2} = \{7,8,9,10,5,6\}$. Por otra parte, para desplazarse de c_2 a c_1 se tiene la ruta C dada por el conjunto $C_{c2,c1} = \{3,2,1\}$.

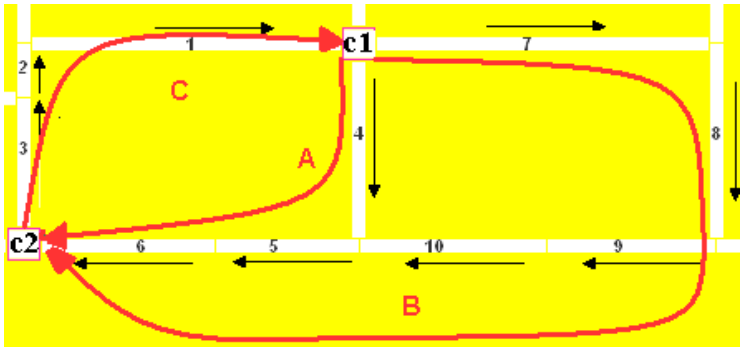


Fig. 3 Rutas disponibles para visitar a los clientes c_1 y c_2

Matriz de rutas

De acuerdo a la definición anterior, un par de clientes tiene asociada una o más rutas disponibles. Esta "*colección"* de rutas la agruparemos en un arreglo matricial que en adelante llamaremos "*matriz de rutas"*. Por ejemplo, los clientes c_1 y c_2 de la Figura 3, tienen las siguientes matrices de rutas:

$$d_1d_2 = \begin{bmatrix} 4 & 5 & 6 & 0 & 0 & 0 \\ 7 & 8 & 9 & 10 & 5 & 6 \end{bmatrix} \qquad d_2d_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

Se agregaron unos ceros "*calles nulas"* a la ruta *A*, esto con el fin de uniformar las matrices ya que por lo general el número de calles será distinto para cada ruta. Observe que las matrices de rutas son distintas incluso en su *orden*, esto se debe a que las rutas son dirigidas y por tanto asimétricas. De manera general, cada renglón de la matriz de rutas representa una ruta disponible y cada columna una calle asociada a dicha ruta (**F**ig. 4).

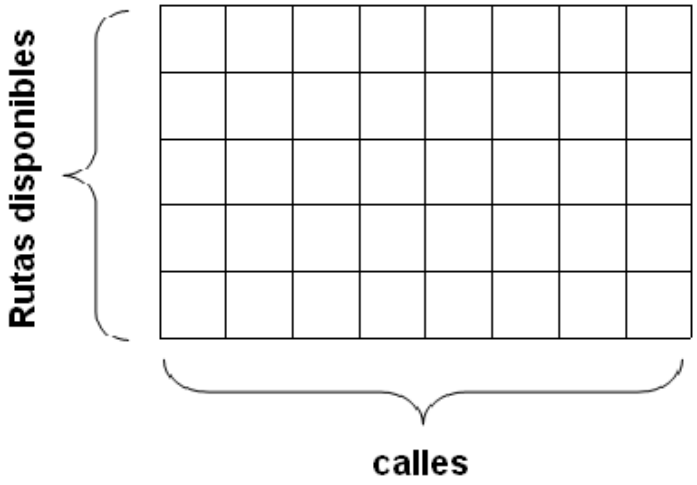


Fig. 4 Matriz de rutas para un par de destinos cualesquiera

Para ejemplificar lo anterior, imagine que un vehículo repartidor de cierto producto necesita realizar un par de entregas a unos clientes (**C1** y **C2**) y regresar al almacén **D**. ¿Cuál sería el recorrido óptimo? La situación se puede plantear como un recorrido de 3 destinos: **D-C1-C2**. El problema radica en determinar no sólo el orden del recorrido, sino por cuáles rutas deberá realizar dicho recorrido. De antemano se sabe que el vehículo debe partir del almacén **D** y regresar al mismo, sin embargo, la decisión crucial es si primero visita al cliente **C1** o mejor al cliente **C2**. Este problema sería muy sencillo de resolver desde el punto de vista del **TSP** clásico con n=3 dado que son pocas las combinaciones que se requieren analizar. Sin embargo, si se añade la característica de distancias dinámicas y se consideran un

mayor número de rutas, el problema se complica ya que no sólo se necesita encontrar un recorrido factible de los clientes sino seleccionar las mejores rutas que conectan dichos clientes considerando los posibles bloqueos.

La **F**igura 5 ilustra esta situación, observe como cada par de clientes se conectan por medio de una *matriz de rutas*. Por ejemplo, los destinos **D** y **C1** se conectan por la matriz **DC1**, en la cual cada renglón representa una posible ruta asociada a esos destinos y las columnas las calles pertenecientes a dicha ruta. Lo mismo ocurre para las demás matrices de rutas (**C1D**, **DC2**, **C2D**, **C1C2**, **C2C1**), con lo que el espacio de búsqueda del problema es mucho más complejo que el de un **TSP**.

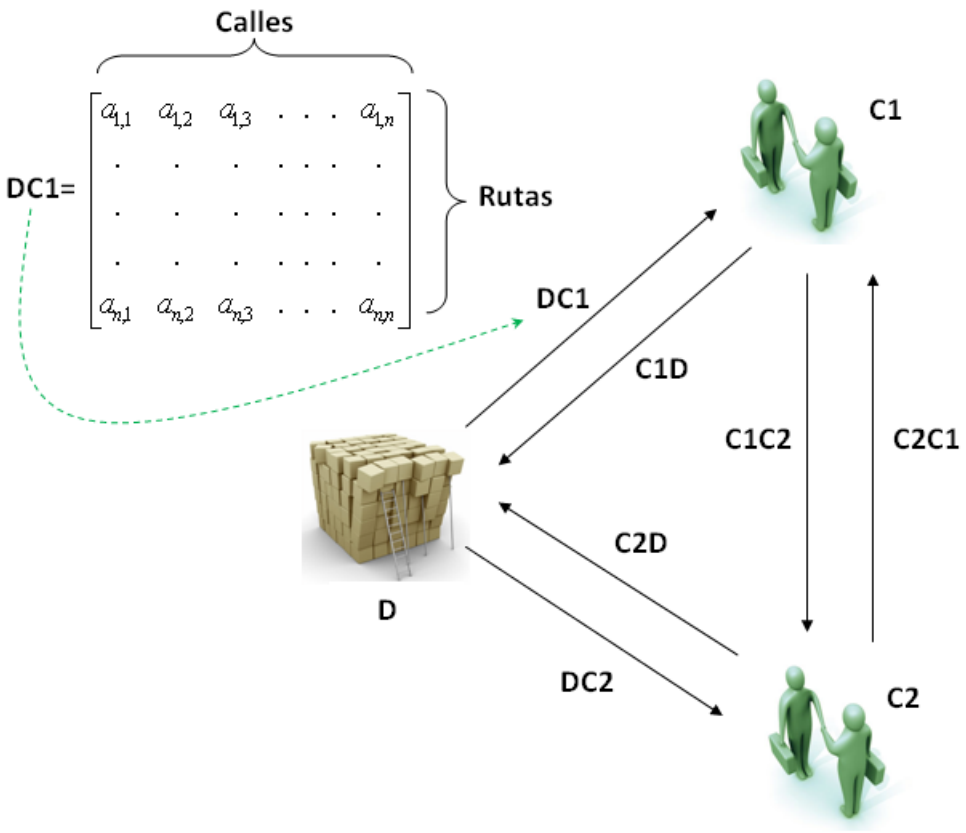
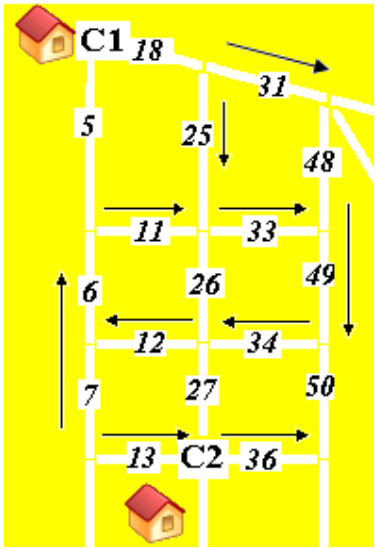


Fig. 5 Problema de ruteo con 3 destinos

- Calle crítica

Una calle crítica es la que pertenece a todas las rutas disponibles de una matriz de rutas. Por ejemplo, sean los clientes **C1** y **C2** con su matriz de rutas asociada $\mathbf{R_{12}}$ la cual contiene todas las rutas disponibles para realizar el viaje de **C1** a **C2** (**F**ig.6).



$$\mathbf{R_{12}}= \begin{pmatrix} 18 & 25 & 26 & 27 & 0 & 0 \\ 18 & 31 & 48 & 49 & 34 & 27 \\ 18 & 25 & 33 & 49 & 34 & 27 \end{pmatrix}$$

Fig.6 Ejemplo de calle crítica

La calle denotada por el dígito ***18***, es crítica dado que está presente en todas las rutas disponibles (renglones) de $\mathbf{R_{12}}$, esto significa que para viajar de **C1** a **C2** forzosamente se debe pasar por dicha calle. Si llegara a presentase un bloqueo en ella será más complicado optimizar el problema o incluso volverse insoluble.

- Matriz de distancias dinámicas **MDD**

Como ya se mencionó, la *distancia dinámica* contempla los bloqueos súbitos en las rutas. Sin embargo, el modelo no sólo debe conocer que rutas están bloqueadas sino en que calles se encuentran dichos bloqueos. Por lo tanto, se deben de almacenar de alguna manera todas las rutas con todas sus calles para poder ubicar exactamente donde se presentan los bloqueos. La **MDD** es una estructura compleja que permite almacenar una gran cantidad de información de manera eficiente. Esta estructura consiste en un arreglo de celdas tipo *"célula"*, en donde cada celda representa una matriz de rutas para un par de clientes en específico. Recuerde que las matrices de rutas son de distinto orden dependiendo de los clientes, por ejemplo, puede ser que un par de clientes tengan muchas rutas disponibles para elegir, y otros pares muy pocas rutas o incluso ninguna. Como ilustración en la **F**igura 7 se puede observar que la celda (1,4) tiene más rutas disponibles que la celda (2,3). Obviamente si el par

de clientes son iguales, las rutas disponibles serán nulas (celda vacía) ya que no existe desplazamiento. Otra manera de visualizar esta estructura, es como una matriz cuyas entradas son a su vez matrices. La **MDD** permitirá calcular más adelante la *distancia dinámica* de las rutas disponibles de cada celda que servirá como parámetro para optimizar el problema.

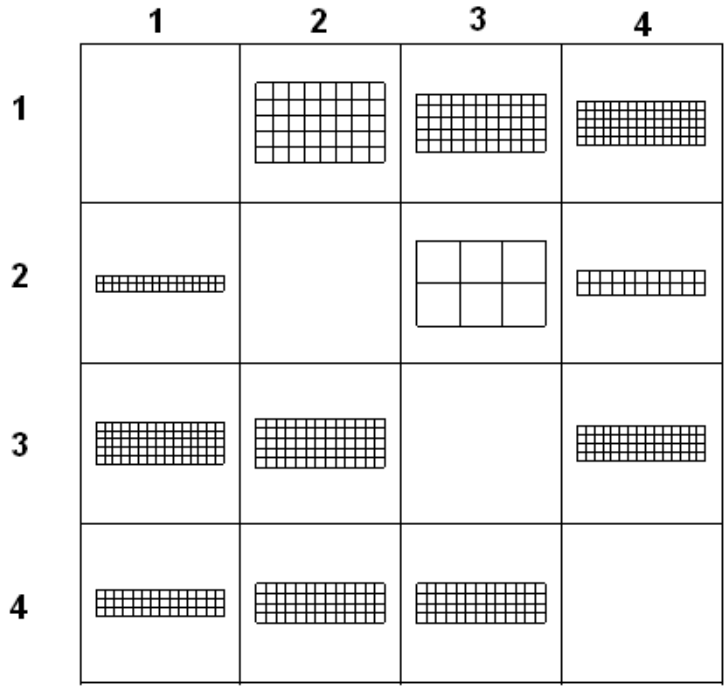


Fig. 7 Matriz de distancias dinámica para almacenar las matrices de rutas.

- Bloqueos

Una calle se considera bloqueada cuando la velocidad promedio del flujo vehicular simulado que circula por dicha calle es prácticamente nula.

- Ruta óptima

La ruta óptima se define como aquélla que conecta a todos los clientes con la menor *distancia dinámica*, y no debe violar las restricciones de los sentidos asignados a las calles (rutas dirigidas) ni repetir clientes.

- Función objetivo

La función objetivo permite calcular la *distancia dinámica* de las rutas para un recorrido en el instante *t*. Recordemos que la distancia dinámica se definió como:

$$d_k(t) = L_k + P * Nb_k(t)$$

Así, la función objetivo está dada por la expresión:

$$F.O. = \sum_{k=1}^{l} L_k + P * Nb_k(t)$$

$$L_k = \sum_{j=1}^{m} \Omega_j$$

Donde:

L_k = Longitud de la ruta k
P = Valor de penalización
$Nb_k(t)$ = Número de bloqueos en la ruta k en el instante t
Ω = Longitud de la calle j
m = número de calles por ruta
l = número de rutas por cliente

Algoritmo híbrido

La ruta óptima se determina y actualiza dinámicamente por medio de un algoritmo híbrido, que consiste de un *Algoritmo Genético* y un *Algoritmo de Simulación de bloqueos*. Una característica relevante de los algoritmos híbridos es que establecen una colaboración entre dos o más algoritmos independientes y explotan los mejores atributos de cada algoritmo para una determinada parte del problema y así proporcionar una solución óptima global [6]. El algoritmo híbrido modela dinámicamente un flujo vehicular con condiciones de bloqueos y busca optimizar rutas alternas para los vehículos. Básicamente el algoritmo mide *la distancia dinámica* de una *población* de rutas por medio de una función objetivo que permite encontrar cuál ruta es la más adecuada para ser recorrida en un instante dado. Si en algún momento se llegaran a presentar bloqueos dentro de la ruta, el algoritmo penaliza su *distancia dinámica* en proporción al número de bloqueos, asignando un valor muy grande a la función objetivo. Posteriormente el algoritmo mide el

desempeño de dicha ruta y en base a un *proceso de selección* estocástico determina si se extingue o sobrevive en la población. Si la ruta llega a sobrevivir, entonces se le aplican ciertos *operadores genéticos* con el fin de mejorar su *distancia dinámica.* Todos los conceptos referentes a los algoritmos genéticos (población, medida del desempeño, proceso de selección, operadores genéticos, etc.) se detallarán más adelante en el capítulo II.

- Algoritmo genético

Se construyó un algoritmo genético con dos poblaciones, la primera permite explorar el espacio de búsqueda, y la segunda sirve para intensificar ciertas regiones de dicho espacio (Figura 8). En esta última se aplicó un operador de cruce que incorpora una estrategia de búsqueda codiciosa con apareo parcial *(Greedy Crossover PMX)*. Las poblaciones del algoritmo tienen las siguientes características:

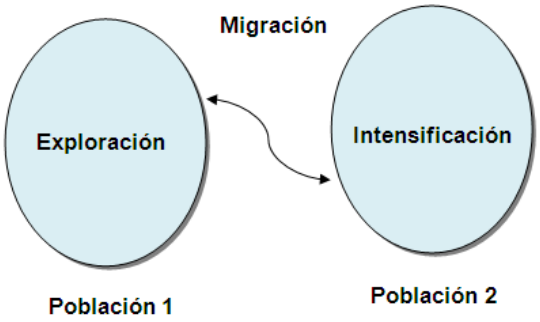


Fig. 8 Algoritmo genético con 2 poblaciones

.

- **Población de exploración**. En esta población se aplicó únicamente el operador de mutación, con el fin de explorar muchas zonas del espacio de búsqueda, y saltar rápidamente de una región a otra. De esta manera las peores soluciones pueden coexistir con las mejores para aumentar la diversidad de soluciones en la población y así, evitar una convergencia prematura.

- **Población de intensificación**. En esta población se aplicó el operador de cruce para intensificar las regiones más prometedoras del espacio de búsqueda y conservar solo las mejores soluciones encontradas por la población de exploración por medio de una búsqueda local codiciosa (*Greedy*). Es decir, sólo los descendientes con mejor desempeño que sus padres sobrevivirán en futuras generaciones. Además, dicha descendencia debe conservar ciertas características de los padres, por ejemplo, no contener ciudades repetidas en el *tour*. Por esta razón se implementó un tipo especial de cruce parcialmente apareado **PMX**, el cual satisface esta condición. El cruce **PMX** será descrito con detalle en el capítulo II.

Las dos poblaciones comparten información emigrando cierta cantidad de individuos cada determinada generación. Esto permite establecer un balance entre la exploración y la intensificación de las soluciones, es decir, se puede variar la cantidad de emigrantes para controlar el rendimiento del algoritmo.

Como ya se mencionó, los algoritmos genéticos son por definición una estrategia de búsqueda "*no informada"*, es decir no conocen el espacio de búsqueda y por tanto no aprovechan toda la información relevante del problema. Por esta razón es necesario añadir cierta información al algoritmo en base al estudio del espacio de búsqueda. En el contexto de nuestro problema, la información relevante es el número de rutas disponibles por cada cliente, y el número de calles que contienen dichas rutas. Toda esta información se proporciona al algoritmo genético por medio de la matriz de distancias dinámicas **MDD**, recuerde que dicha matriz es una estructura compleja que almacena una gran cantidad de información, como son rutas disponibles y calles.

- Algoritmo de simulación de bloqueos

Se diseñó un simulador de condiciones de tráfico, para generar bloqueos por situaciones imprevistas en un día normal de tráfico, como puede ser algún accidente, o desvíos por obras de mantenimiento, y de esta manera someter al algoritmo genético a diferentes escenarios. La simulación de bloqueos se basó en el modelo del conductor inteligente **IDM**, el cual describe el comportamiento de los conductores en un flujo vehicular, así como su reacción ante bloqueos inesperados. Este modelo se describirá con más detalle en el capítulo III.

Capítulo II
Algoritmos Genéticos

``Es extraño pero cierto; porque la verdad siempre es extraña, más extraña que la ficción ‘‘
Lord Byron

El principio de *“supervivencia del más apto”* propuesto por Charles Darwin en 1859, ha sido utilizado como una idea importante para el desarrollo de la computación evolutiva, una pregunta relevante en este tema es ¿Cómo produce la evolución organismos cada vez más aptos en ambientes extremadamente complejos?

La *teoría de la selección natural* establece que las plantas y los animales que existen hoy en día, fueron el resultado de millones de años de adaptación a las cambiantes condiciones del ambiente. En algún momento, un número de diversos organismos coexistieron y compitieron entre ellos por los mismos recursos de su ecosistema. Una suposición en esta teoría, es que los organismos que salieron triunfantes en esta lucha, fueron los más capaces y por tanto, sus descendientes resultaron numerosos en el futuro, y con el tiempo la población entera del ecosistema generó organismos, que, en promedio, son más aptos que los de generaciones previas, ya que poseen características las cuales tienden a mantener su supervivencia [6].

Cualquier organismo vivo consiste de células que contienen un conjunto único de uno o más *cromosomas* formando cadenas de ADN, cada *cromosoma* puede ser dividido en *genes*, y cada *gen* contiene una proteína particular [13]. Cada una de las posibles configuraciones de las proteínas forman lo que se denominan los *alelos*, y cada *gen* se localiza en una posición particular (*locus*) del cromosoma [12]. Existen organismos que tienen varios cromosomas en cada célula formando lo que se denomina *genoma*; el término *genotipo* se refiere a un conjunto particular de *genes* del *genoma*. Los organismos cuyos *cromosomas* están dispuestos en pares se denominan *diploides*, aquellos que no están dispuestos en pares se denominan *haploides*; la mayoría de los organismos con capacidad reproductora son *diploides*, incluidos los seres humanos que tienen 23 pares de cromosomas en cada célula. Durante la reproducción sexual se produce el proceso de "recombinación" o *cruce* en el que cada padre proporciona determinados genes que son intercambiados entre cada par de cromosomas formando un gameto con un único cromosoma; los gametos de ambos padres se emparejan creando un conjunto completo de cromosomas diploides. En la reproducción sexual haploide los genes son intercambiados entre los dos padres mediante una única cadena de cromosomas. Durante este proceso de reproducción los hijos están sujetos a mutaciones en las cuales los elementos más elementales de un cromosoma son intercambiados de padres a hijos provocando frecuentemente errores de copia. La "*aptitud*" de un

organismo se define como la probabilidad que tendrán de vivir y reproducirse o bien como el número posible de hijos que puede tener.

En el contexto de los algoritmos genéticos, el término "*cromosoma"* se aplica a la *solución candidata* a un problema que generalmente es codificada en un cierto alfabeto, por ejemplo el binario (ceros y unos) [12]. El término "*gen*" se refiere a un parámetro particular de la solución candidata y un "*alelo*" se refiere al valor de dicho parámetro (**F**ig.9) [12]. El "*operador cruce*" (*crossover*) consiste en el intercambio de material genético entre dos padres *haploides* de un único cromosoma y el "*operador mutación*" consiste en el cambio aleatorio de un gen en una posición escogida de forma aleatoria [18].

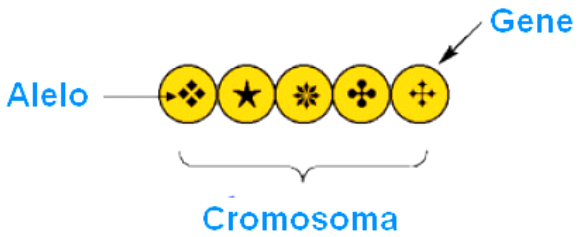


Fig.9 Codificación de soluciones en forma de cromosoma

Naturaleza de un algoritmo genético

Un algoritmo genético es una estrategia de búsqueda inspirada en los mecanismos de adaptación de los sistemas biológicos para resolver problemas. La mayoría de los **AG** constan básicamente de las siguientes partes: una población de soluciones codificadas en cromosomas, un método de selección, y un conjunto de operadores genéticos como el cruce y la mutación [7] [11].

Dado un problema específico a resolver, los valores de entrada del **AG** es un conjunto de "*soluciones potenciales*" del problema, codificadas de alguna manera, y una métrica llamada función de desempeño que permite evaluar cuantitativamente cada solución, las cuales en adelante se denominan *individuos.* En efecto, un proceso evolutivo que ocurre en una población de individuos corresponde con la búsqueda del óptimo en el conjunto de las *soluciones potenciales*.

Considerando que un **AG** actúa como un método de búsqueda sobre una población, ésta sufre la simulación de una evolución: en cada generación los individuos

relativamente *"buenos"* se reproducen, mientras que los *"malos"* se extinguen. Luego se realizan múltiples copias de los individuos, pero éstas no son perfectas; se introducen cambios aleatorios durante el proceso de duplicado mediante operadores genéticos que son los encargados de introducir nuevos elementos (descendientes) en la población. Durante este proceso, dos son los operadores genéticos que entran en funcionamiento: el cruce y la mutación.

Posteriormente, esta descendencia forma un nuevo acervo de soluciones candidatas (*mate pool*), que son sometidas a una ronda de evaluación de desempeño. Los individuos que no han mejorado con los cambios en su código son eliminados, no obstante, las variaciones aleatorias introducidas en la población pueden favorecer algunos individuos, convirtiéndolos en mejores soluciones del problema, más completas o más eficientes. De nuevo, se seleccionan y se replican estos individuos vencedores hacia la siguiente generación introduciendo cambios aleatorios [5].

Este mecanismo se repite durante generaciones hasta que se alcanza una cierta condición de parada. Es decir, el algoritmo finaliza cuando se ha ejecutado un número determinado de iteraciones prefijado de antemano, ó cuando se encuentra el óptimo (por ejemplo, después de un cierto número de generaciones, el programa detecta al mejor individuo, que representa la mejor solución al problema).

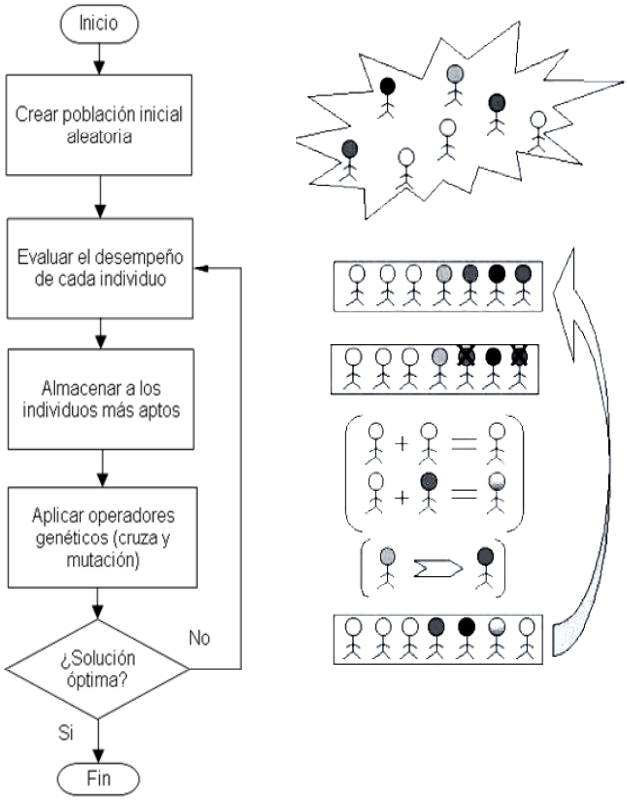


Fig. 10 Diagrama de flujo de un algoritmo genético.

La expectativa es que el desempeño medio de la población se incrementará en cada generación y, por tanto, repitiendo este proceso cientos o miles de veces, pueden hallarse soluciones muy precisas del problema. La **F**igura 10 muestra el proceso iterativo de un **AG**.

Tipos de codificación

Para utilizar un **AG** en un problema de optimización, es necesario codificar el conjunto de parámetros del problema de alguna manera. Por lo general dicha codificación se realiza con arreglos binarios, sin embargo, en general la codificación puede consistir de arreglos de longitud finita de cierto alfabeto (por ejemplo, números enteros, reales, etc.) [8].

Por ejemplo, para maximizar la función $f(x) = x^2$ en el intervalo $[0,31]$ se puede codificar a la variable *x*, como un conjunto de posiciones (*s*) de un arreglo de interruptores en una caja negra [**F**igura 11], luego el **AG** procesa "*s*" hasta encontrar el valor óptimo de $f(x)$.

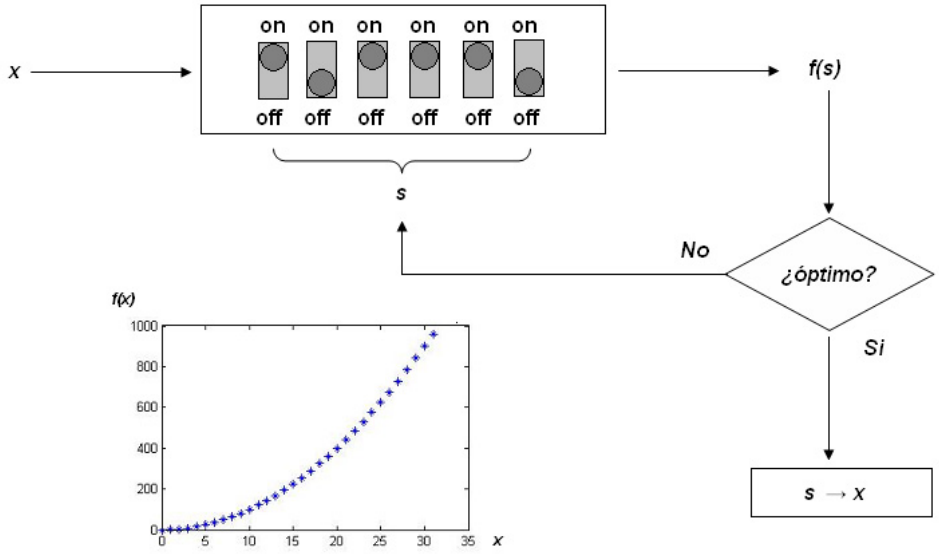


Fig. 11 Evaluación de una solución codificada

Por su sencillez, la codificación basada en el alfabeto binario (*codificación binaria*) [**F**igura 12] es la más utilizada, aunque para ejemplos concretos se pueden utilizar otro tipo de codificaciones dependiendo de la estructura del problema a tratar.

Cromosoma 1	1 1 0 1 0 0 0 1 1 0 1 0
Cromosoma 2	0 1 1 1 1 1 1 1 1 1 0 0

Fig. 12 Codificación binaria

Por ejemplo, pensemos en un problema donde se requiere codificar una estructura que encierre información bidimensional – es el caso de una matriz:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Se podría representar fácilmente por una cadena de dieciséis genes, concatenando sus filas (o sus columnas):

$$(a_{11}, a_{12}, a_{13}, a_{14}, a_{21}, a_{22}, a_{23}, a_{24}, a_{31}, a_{32}, a_{33}, a_{34}, a_{41}, a_{42}, a_{43}, a_{44})$$

Sin embargo, esta codificación ignora el hecho de que a_{11} y a_{21} eran adyacentes en la matriz; por el contrario, a_{24} y a_{31} lo son ahora, cuando antes estaban distantes.

La codificación de imágenes es otro ejemplo en el que se pone de manifiesto la importancia de establecer una representación apropiada de los datos, que no pierda la correlación existente entre los genes adyacentes; es decir, que no destruya información importante para el problema. No obstante que la codificación binaria ha sido muy utilizada en **AGs**, existen formas de codificación alternativas como el asignar números enteros, ó reales, a los parámetros del problema (**F**ig.13) [10].

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

Fig. 13 Codificación entera

La elección de la codificación de las posibles soluciones, mediante cadenas (en cuanto a la longitud de las mismas o los valores de los alelos) es fundamental en la eficiencia del algoritmo Existen algunas ventajas prácticas en la utilización de valores reales para codificar los parámetros de un **AG**; por ejemplo, mejora la eficiencia ya que no es necesario decodificar cada parámetro para evaluar la función objetivo. Se utiliza menos memoria y pueden emplearse arreglos de variables de punto flotante para representar un cromosoma [10].

Función Objetivo y Función de Aptitud

La *función objetivo* proporciona una medida del *desempeño* del sistema asociado a cierto individuo en la población. La *función de aptitud* se utiliza para convertir los valores de la función objetivo en valores de *aptitud* de cada individuo. La aptitud es una medida de la calidad de la solución, un valor de aptitud alto corresponde a un desempeño alto (en un problema de maximización) o a un desempeño bajo (en un problema de minimización). Los valores de la función de aptitud se utilizan como parámetro para seleccionar a los individuos que se reproducirán en la siguiente generación [18].

Algoritmos de Selección

Un algoritmo genético puede utilizar muchas técnicas diferentes para seleccionar a los individuos que deben copiarse hacia la siguiente generación, algunos de estos métodos son mutuamente exclusivos, pero otros pueden combinarse. Los algoritmos de selección, asignan de manera probabilística un número entero de copias a un individuo de acuerdo a su aptitud. A continuación se describen algunos mecanismos de selección [14].

- *Elitismo*

Se selecciona al individuo (o los individuos) con mayor aptitud de cada generación, es decir se fuerza a que el mejor individuo de la población en el tiempo *t*, sea seleccionado como padre.

- *Proporcional a la aptitud*

La función de selección de padres más utilizada es la denominada función de selección proporcional a la aptitud, la cual consiste en seleccionar a los cromosomas de la población de acuerdo a un índice de aptitud. Se asignan probabilidades de selección p_i a los diferentes individuos según su valor de aptitud relativa q_i dividido por la aptitud total de la población. Entonces la probabilidad de selección de un individuo p_i está dada por la expresión:

$$p_i = \frac{q_i}{\sum_{i=1}^{n} q_i}$$

Sin embargo, el uso de la aptitud relativa para asignar copias a cada individuo, puede propiciar una convergencia prematura del **AG**: individuos con valores grandes de aptitud relativa en las primeras generaciones pueden dominar las generaciones

sucesivas, lo que significa que el algoritmo puede quedar atrapado en óptimos locales [8].

- *Ranking lineal*

La población de *N* individuos se ordenan de acuerdo a la aptitud de cada uno y se limita el número de copias (*presión selectiva*) que se espera de cada individuo usando la siguiente expresión:

$$p_i = 2 - MAX + 2(MAX - 1)\left[\frac{x_i - 1}{Nind - 1}\right]$$

Donde:

x_i : Es la posición del individuo *i*

$1 \le MAX \le 2$: Es el número máximo de copias que puede recibir un individuo (Presión selectiva).

Nind: Es el número de individuos de la población.

La ventaja de este método es que puede evitar que individuos muy aptos dominen al principio a expensas de los menos aptos, lo que reduciría la diversidad genética de la población y podría obstaculizar la búsqueda de una solución aceptable [8].

- *Rueda de ruleta RWS*

El método por rueda de ruleta ("*RWS"* por sus siglas en inglés "*Roulette Wheel Selection"*) es otra forma de selección proporcional a la aptitud en la que la probabilidad para que un individuo sea seleccionado, depende de la diferencia entre su valor de aptitud y la de sus competidores. Este mecanismo de selección puede representarse como un juego de ruleta [**F**igura 14]. Por ejemplo, a cada individuo le corresponde una sección de la ruleta, pero a los más aptos se les asigna secciones mayores que las de los menos aptos. Los individuos son escogidos a partir de marcadores, igualmente espaciados y con comienzo aleatorio.

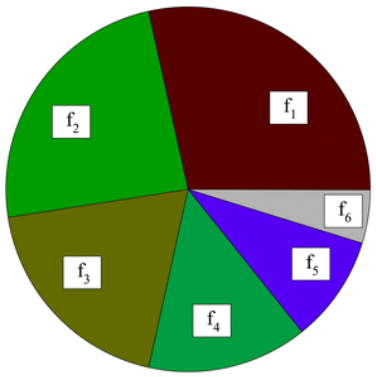


Fig. 14 Selección por rueda de ruleta

- *Torneo*

Constituye un procedimiento de selección de padres muy extendido, la idea consiste en escoger al azar un número de individuos de la población (*tamaño del torneo*), y

luego seleccionar el mejor individuo de este grupo, y repetir el proceso hasta que el número de individuos seleccionados coincida con el tamaño de la población.

.

Operadores genéticos

Una vez seleccionados los individuos, éstos deben ser alterados de forma aleatoria con la esperanza de mejorar su aptitud para la siguiente generación. Existen dos operadores básicos para llevar esto a cabo. El primero y más sencillo se llama *mutación*. De la misma manera que una mutación en los seres vivos cambia un gen por otro, una mutación en un algoritmo genético también causa pequeñas alteraciones en puntos concretos del código de un individuo [7].

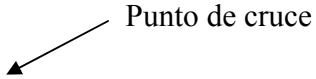
La mutación se considera un operador fundamental ya que proporciona un pequeño elemento de aleatoriedad en la vecindad (*entorno*) de los individuos de la población y es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones [9].

Para ilustrar la mutación considere una codificación binaria, entonces, el operador de mutación se aplica bit a bit con una probabilidad p_m. Se genera un número aleatorio r uniforme entre 0 y 1, si r es menor que p_m el bit muta su valor de 0 a 1 o de 1 a 0, según el caso [**F**igura 15]. Esta transformación se utiliza con el fin de que el **AG** tenga capacidad de exploración del espacio de búsqueda y evite quedar atrapado en óptimos locales. Los individuos resultantes después de la mutación pasan a formar parte de la nueva generación.

Fig. 15 Mutación de una cadena binaria

El segundo operador se llama *cruce*, el cual consiste en elegir a dos individuos para que intercambien segmentos de su código, produciendo una ``descendencia" cuyos individuos son combinaciones de sus padres. Este proceso pretende simular el proceso análogo de la recombinación que se da en los cromosomas durante la reproducción sexual [7].

El cruce de un punto funciona seleccionando una posición aleatoria dentro del cromosoma y se intercambian los bits de cola derecha, formando así dos nuevos descendientes (uno para cada padre) [Figura 16].



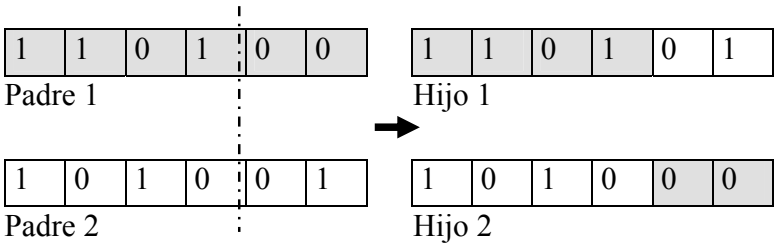


Fig.16 Cruce de 2 cadenas binarias

Esquemas de reemplazo

Conforme se van creando nuevos individuos debe utilizarse algún mecanismo que mantenga el tamaño de la población, de forma que algunos o todos los individuos de la población sean sustituidos por los nuevos individuos generados. Fundamentalmente se utilizan tres procedimientos para realizar esta sustitución:

- *Reemplazo generacional*, en este esquema la población completa es renovada, se asemeja a la forma de reproducción de los insectos, donde una generación pone huevos, se aleja geográficamente o muere y es substituida por una nueva. Sin embargo, bajo estas circunstancias no se garantiza que el mejor individuo permanezca en la población. Por esta razón, en problemas de optimización se suele utilizar un enfoque elitista, de manera que los *k* mejores individuos (*k-elitismo*) se mantengan de una generación a la siguiente en la población

- *Estado estacionario* (*steady-state*), en este procedimiento sólo unos pocos individuos (típicamente uno o dos) de la población son sustituidos. El estado estacionario utiliza el esquema generacional de los mamíferos, donde coexisten padres y sus descendientes, permitiendo que los hijos sean educados por sus progenitores, pero también que a la larga se genere competencia entre ellos. En este modelo, no sólo se deben seleccionar los dos individuos a ser padres, si no también cuales de la población anterior serán eliminados, para dar espacio a los descendientes.

- *Renovación gradual*, generaliza los anteriores, de manera que sólo una fracción de la población (*gap*) es sustituida. En general, cuanto menor sea la fracción de la población que se renueva, más rápida puede ser la convergencia del algoritmo. Aunque esto puede resultar ventajoso, es necesario cuidar que no se trate de una convergencia prematura.

Fundamentos matemáticos de un algoritmo genético

A fin de explicar cómo trabaja un **AG** introduciremos algunas definiciones que permitan entender el *"Teorema de Schema"* desarrollado por John Holland en 1975 [1]. Dicho teorema analiza el efecto de los procesos de selección, cruce y mutación sobre un ente matemático llamado *schema* (en plural *schemata*).

Definición de schema

Un *schema* es una plantilla de patrones que representa un subconjunto de cromosomas con secciones similares, es decir, con correspondencias en ciertas posiciones. Por ejemplo, si consideramos el alfabeto binario V = {0,1} y agregamos un tercer elemento "*" (no importa su valor). Podemos entonces crear arreglos *schemata.*

Un *schema* puede verse como un mecanismo de correspondencia de patrones. Un cierto *schema* manifiesta un patrón si los unos y ceros en el patrón corresponden para todas las posiciones.

Considere, el *schema* H=1****1, éste representa el conjunto de todas las cadenas de 6 bits que comienzan y terminan con 1. Las cadenas de bits que corresponden a una determinada plantilla de similitud (por ejemplo, 100111 y 110011) se llaman "*instancias"* de H.

El "*" es un elemento de notación que permite representar todas las posibles similitudes entre arreglos de una cierta longitud y alfabeto.

El número total de *schemata (Ns)* posibles para arreglos de longitud *l* sobre un alfabeto de cardinalidad *k* está dado por

$$Ns = (k+1)^l .$$

Por ejemplo, para k=2, l=5; *Ns*=243

Los schemata nos permiten evaluar la cantidad de información que se incorpora en el proceso de búsqueda (optimización) si incluimos las similitudes entre los arreglos de una población.

Cualquier arreglo de bits de longitud l es una instancia de 2^l *schemata* diferentes. Por ejemplo, la cadena 11 es una instancia de ** (todas las 4 posibles cadenas de longitud 2), *1,1*, y 11 (el schema que contiene sólo una cadena, 11).

Del mismo modo, cualquier población de n cadenas contiene instancias entre 2^l y $n2^l$. Esto significa que, en una determinada generación mientras el AG está explícitamente evaluando la aptitud de las n cadenas en la población, también estima implícitamente la aptitud promedio de un número mucho más grande de *schemas*, donde la aptitud promedio de un schema se define por la aptitud promedio de todas las posibles instancias del schema. Por ejemplo, en una población generada de manera aleatoria de n cadenas, en promedio la mitad de las cadenas serán instancias de 1***…* y la otra mitad instancias de 0***---*. Las evaluaciones de aproximadamente n/2 cadenas que son instancias de 1***…* darán un estimado de la aptitud promedio del schema (es un estimado debido a que las instancias evaluadas en un tamaño de población típica existen sólo algunas muestras de todas las posibles instancias).

Consideremos el arreglo de 5 bits: A_1: 1 1 1 1 1. Este arreglo es uno de entre 25 schemata, ya que cualquiera de las 5 posiciones puede ser ocupada por su valor o por un *.

Por ejemplo, para arreglos binarios con l=5 y n=4 tenemos entre 32 y 128 *schemata* diferentes. Nótese como con sólo 4 arreglos podemos representar la mitad de los 243 *schemata* que existen para l=5.

Para aclarar las definiciones anteriores, sean los arreglos binarios (k=2) con longitud l=2, con estos valores se pueden formar en total 4 arreglos (00, 01,10,11), por lo tanto el número de schemata para dichos arreglos está dado por:

$$Ns = (k+1)^l = (2+1)^2 = 9$$

Y el número de schemata por arreglo será:

$$Ns / arreglo = (k)^l = (2)^2 = 4$$

Esta última expresión indica que existen 4 schemata por cada arreglo, dado que se tienen 4 arreglos, el número total de schemata sería 16. Esta aparente contradicción se aclara visualizando la siguiente tabla, observe que existen schemata repetidos (tachados) en los diferentes arreglos dando un total de 9 como era de esperarse.

Arreglo	Schemata	Ns
0 0	* * * 0 0 * 0 0	4/9
0 1	~~* *~~ * 1 ~~0 *~~ 0 1	2/9
1 0	~~* *~~ 1 * ~~* 0~~ 1 0	2/9
1 1	~~* *~~ ~~* 1~~ ~~1 *~~ 1 1	1/9

Tabla.1 Enumeración de *schemata* representados por arreglos de 2 bits.

Propiedades de schemata

- *Orden O(H)*

Es el número de posiciones fijas en un *schemata* (diferentes de *).

Por ejemplo, sean los *schemata*:

H_1 = 0 1 1 * 1 * *

H_2 = 0 * * * * * *

0 (H_1) = 4

0 (H_2) = 1

- *Longitud de definición δ(H)*

Es la distancia entre la primera y la última posición fija de un *schema*.

Por ejemplo:

$\delta(H_1)$ = 5-1 = 4;

$\delta(H_2)$ = 0

Teorema de Schema

Considerando las definiciones anteriores, ya se está listo para analizar los efectos de los operadores genéticos sobre *schemata*. En este apartado se analizan los efectos de la selección, cruce y mutación por separado, para finalmente conformar el teorema de schema, el cual nos ayuda a predecir el número de copias que un determinado schema recibirá en el proceso generacional.

- *Efectos de la selección*

Holland propuso una fórmula para predecir el número de copias de *schemata* que tendrá en la siguiente generación después de aplicar reproducción proporcional al desempeño, cruce y mutación.

Sea *m(H, t)* el número de instancias de *H* en un ciclo *t* del algoritmo con *n* individuos en la población. El número de copias de *schemata* en la siguiente generación está dado por la siguiente expresión:

$$m(H,t+1) = m(H,t)\cdot n \cdot \frac{f(H)}{\sum f_j} \quad \ldots\ldots\ldots\ldots\ldots (1)$$

Si definimos el desempeño promedio de la población como:

$$\bar{f} = \frac{\sum_{j=1}^{n} f_j}{n}.$$

Podemos reescribir la ecuación (1) de la siguiente manera:

$$m(H,t+1) = m(H,t)\cdot \frac{f(H)}{\bar{f}} \quad \ldots\ldots\ldots\ldots (2)$$

La interpretación de esta ecuación refleja el efecto de la selección en los **AGs**:

1) *Schemata* con desempeño por encima de la media reciben un número creciente de copias en la siguiente generación.
2) *Schemata* con desempeño por debajo de la media tienden a desaparecer.

- *Efectos del operador cruce*

Sea p_d la probabilidad de extinción de un *schema H* por el cruce de un punto:

$$P_d = \frac{\delta(H)}{l-1}$$

Por lo tanto,

$$P_s = 1 - \frac{\delta(H\ )}{l-1}$$

es la probabilidad de supervivencia de esa plantilla.

Siendo p_c la tasa de aplicación del crossover y considerando que el par progenitor de un cromosoma puede recuperar parte de un patrón destruido por el crossover, tenemos la siguiente desigualdad:

$$P_s \geq 1 - P_c \cdot \frac{\delta(H)}{l-1}$$

Esto significa que *schemata* cortos tienen mayor probabilidad de supervivencia después del crossover.

El efecto combinado de la reproducción y el crossover esta dado entonces por:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_c \cdot \frac{\delta(H)}{l-1} \right] \ldots\ldots\ldots\ldots (3)$$

- *Efectos del operador mutación*

Consideremos ahora una probabilidad de mutación P_m. La probabilidad de sobrevivir de un alelo particular es de 1-P_m y la probabilidad de sobrevivencia en un *schema* está dada por:

$$P_{sm} = \left(1 - P_m\right)^{O(H)}$$

Si $P_m << 1$ podemos aproximar P_{sm} como:

$$P_{sm} = 1 - O(H)P_m$$

Esto significa que *schemata* de orden bajo tienen mayor probabilidad de sobrevivir por la mutación. El número esperado de copias de un *schema* H tomando en cuenta la aplicación de reproducción, crossover y mutación está dado por el *Teorema de schema:*

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_c \cdot \frac{\delta(H)}{l-1} - O(H)P_m \right]$$

Un primera conclusión derivada de la ecuación anterior es que aquéllos *schemata* con un desempeño superior al promedio de la población, tendrán mayor posibilidad de aumentar su número de copias en la siguiente generación. Así, *schemata* cortos, de bajo orden, y adaptación por encima de la media, son favorecidos en generación futuras. Esto se conoce como ***"hipótesis de los bloques de construcción",*** de acuerdo con esta hipótesis, el algoritmo genético obtiene mejores soluciones a partir de la combinación de información de pequeños segmentos de individuos correspondientes a esos bloques constructores. Al teorema de *schema* se le han hecho fuertes críticas con respecto a las hipótesis y sus conclusiones. Algunos resultados [25] muestran que los algoritmos genéticos, que habían sido propuestos como un método de búsqueda mediante caja negra, pierden efectividad si la representación de los individuos y los operadores genéticos tienen poco que ver con la estructura de las soluciones del problema.

A continuación se presenta un ejemplo para comprender y analizar el funcionamiento de un **AG**. Suponga que desea maximizar la función $f(x) = x^2$ en el intervalo $[0,31]$ (Ver la **T**abla 2). Los pasos para resolver este problema son los siguientes:

Paso 1: La primera etapa para resolver un problema utilizando AGs es codificar las posibles soluciones en cromosomas. En este caso se utilizó una codificación binaria

de cinco bits, así los números 0(00000) y 31(11111) pueden obtenerse con este rango. La población inicial fue generada aleatoriamente mediante cuatro repeticiones de cinco volados, donde cara=1 y cruz=0. Aquí se ilustra con una población de tamaño 4, por supuesto que se puede elegir cualquier tamaño de población dependiendo solamente de los requerimientos del problema.

Arreglo	*Población inicial (aleatoria)*	*x*	*f(x)=x²*	*Probabilidad de selección* $\frac{f_i}{\sum f}$	*Copias esperadas* $\frac{f_i}{\bar{f}}$	*Copias actuales de la rueda de ruleta*
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Suma			1170	1.00	4.00	4.0
Prom.			293	0.25	1.00	1.0
Max.			576	0.49	1.97	2.0

Tabla 2. Algoritmo genético paso por paso

Mating Pool (Después de la reproducción)	*Selección de padres (seleccionados al azar)*	*Lugares para el cruce (seleccionados al azar)*	*Nueva población*	*Valor de x*	*f(x)=x²*
0 1 1 0 1	2	4	0 1 1 0 0	12	144
1 1 0 0 0	2	4	1 1 0 0 1	25	625
1 1 0 0 0	4	2	1 1 0 1 1	27	729
1 0 0 1 1	3	2	1 0 0 0 0	16	256
					1754
					439
					729

Tabla 2. (continuación)

Paso 2: Se decodifican las soluciones a números enteros sin signo para obtener el valor de x. Por ejemplo, considerando la cadena 1 se tiene:

$01101 = 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$
$= 0 + 8 + 4 + 0 + 1$
$= 13$

Luego se obtienen los valores decodificados para todas las cadenas.

Paso 3: Calcular el *"fitness"* de la función objetivo. Éste se obtiene simplemente elevando al cuadrado los valores de "*x*", dado que la función objetivo es $f(x) = x^2$. Por ejemplo, cuando *x*=13, el valor de fitness es:

$$f(x) = x^2 = (13)^2 = 169$$

para $x = 24$, $f(x) = x^2 = (24)^2 = 576$

y así sucesivamente para la población entera.

Paso 4: Calcular la probabilidad de selección,

$$prob_i = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i}$$

Donde *n* es el tamaño de la población.
f(x) representa el valor de *fitness* correspondiente a un individuo en particular de la población
Σf(x) Expresa la sumatoria de todos los valores de *fitness* de la población entera.

Por ejemplo, sea la cadena 1,

$$Fitness\ f(x) = 169$$
$$\Sigma f(x) = 1170$$

La probabilidad de que la cadena 1 ocurra está dada por,

$$P_1 = 169/1170 = 0.14$$

Es decir, el porcentaje de probabilidad para que ocurra es 14 %.

Se realizan las mismas operaciones para todas las cadenas, fácilmente se puede apreciar que la sumatoria de todas las probabilidades de selección debe ser 1.

Paso 5: Se calculan el número esperado de copias, el cual se obtiene de la siguiente manera:

$$\textit{Número de copias esperadas} = \frac{f(x)_i}{(promf(x))_i}$$

Donde $(promf(x))_i = \dfrac{\sum_{i=1}^{n} f(x)_i}{n}$

Por ejemplo, para la cadena 1 se tiene,

Número de copias esperadas = *Fitness*/*promedio* = 169/293 = 0.58

Del mismo modo se calculan el número de copias esperadas para la población entera. Las copias esperadas proporcionan una idea de qué individuos pasaran a la siguiente generación para ser procesados.

Paso 6: Se seleccionan a los individuos por medio del método de rueda de ruleta **F**ig. 17. La rueda de ruleta representa el 100% y las probabilidades de selección calculadas en el paso 4 se utilizan como indicadores que corresponden dentro de la rueda de ruleta:
La cadena 1 ocupa el 14 %, así que tiene oportunidad de ocurrir al menos una vez, por lo tanto su número de copias actuales puede ser igual a 1. La cadena 2 ocupa el 49% de la rueda de ruleta, por lo que tiene oportunidad de ser seleccionada dos veces, así que su número actual de copias puede ser 2. Sin embargo, la cadena 3 tiene un porcentaje muy bajo de probabilidad del 6%, por tanto, su ocurrencia para el siguiente ciclo es muy pobre. Como resultado, su copia actual es 0. La cadena 4 con 31% tiene al menos una oportunidad para ocurrir mientras la rueda de ruleta gira, en consecuencia su cuenta actual es 1. Los valores del número de copias esperadas se muestran en la **T**abla 2 (ver pag.37).

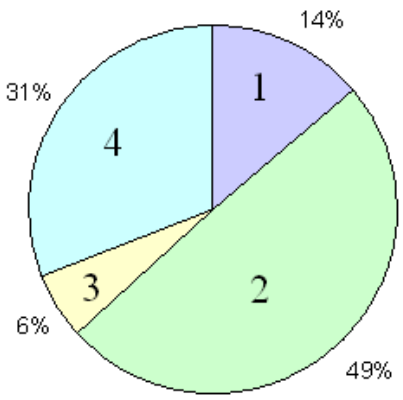


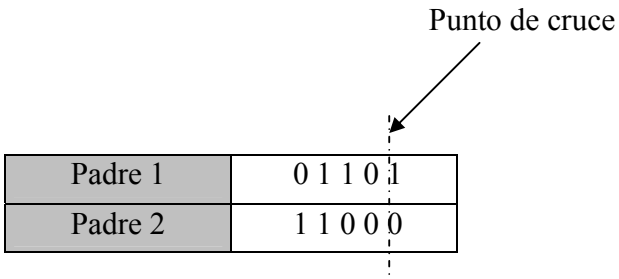
Fig. 17 Selección por rueda de ruleta

Paso 7: Se almacenan las cadenas a las que se les aplicarán los operadores genéticos, a este conjunto de cadenas almacenadas se le conoce como el "*mating pool"* el cual contiene el número de copias actuales como se observa en la **T**abla 2.

El número de copias actuales para la cadena 1 es 1, por lo tanto ocurre una vez en el *mating pool*. El número de copias actual para la cadena 2 es 2, por lo que sucede dos veces en el *mating pool*. De manera similar, las copias actuales de la cadena 4 es 1, y acontece una vez en el *mating pool*. De esta manera es como se forma el *mating pool*. Nótese en la **T**abla 2 que la cadena 3 se extinguió debido a que tenía una probabilidad de selección muy baja.

Paso 8: Se seleccionan de manera aleatoria los padres a los cuales se les aplica el operador de cruce. Para ello, se deben elegir los puntos para el cruce. En este ejemplo se utiliza el cruce en un solo punto. La probabilidad de cruce se asumió de P_c=1.

Por ejemplo:



Padre 1	0 1 1 0 1
Padre 2	1 1 0 0 0

La descendencia producida es,

Hijo 1	0 1 1 0 0
Hijo 2	1 1 0 0 1

De manera similar el operador de cruce se implementa en las siguientes cadenas.

Paso 9: Una vez efectuadas las operaciones de cruce, la descendencia produce una nueva población y por lo tanto nuevos valores de "*x*" se decodifican para luego calcular su *fitness*.

Procesamiento de Schemata

A continuación se analiza como procesa los *schemata* el algoritmo genético. Primero obtendremos una tabla antes de la reproducción. Considere los siguientes schemata:

H_1: 1 * * * *

H_2: * 1 0 * *

H_3: 1 * * * 0

De acuerdo a la **T**abla 2 (pag. 39) el *schema* H_1 aparece en las cadenas 2 y 4, el *schema* H_2 en las cadenas 2 y 3, y el *schema* H_3 está presente en la cadena 2. Si se calcula el desempeño promedio de las cadenas se obtiene la **T**abla 3.

Por ejemplo, de acuerdo a la **T**abla 3 para el *schema* H_1 se tiene que:

$$f(H) = \frac{576 + 361}{2} = 469$$

Del mismo modo se calcula el desempeño promedio para todos los *schemata* restantes.

Schema	***Cadenas***	***Desempeño promedio f(H)***
H_1 : 1 * * * *	2, 4	469
H_2 : * 1 0 * *	2, 3	320
H_3 : 1 * * * 0	2	576

Tabla 3. Procesamiento de *schemata* antes de la reproducción

De acuerdo al teorema de *schema*, el número esperado de copias después de la reproducción se obtiene por la siguiente expresión:

$$m(H, t+1) = m(H, t) \cdot \frac{f(H)}{\bar{f}}$$

Por ejemplo para el *schema* H_1 estos números son:

$$m(H_1, 1) = 2$$

$$m(H_1, 2) = 2 \cdot \frac{469}{293} = 3.20$$

Así para todos los *schemata*, ver **T**abla 4.

Número esperado de copias	*Número de copias*	*Cadenas*
H_1 : 3.20	3	2,3,4
H_2 : 2.18	2	2,3
H_3 : 1.97	2	2,3

Tabla 4. Procesamiento de schemata después de la reproducción

Por último, considerando el operador de cruce, el número de copias de *schemata* esperadas se calcula por:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_c \cdot \frac{\delta(H)}{l-1} \right]$$

Por ejemplo para el *schema* H_1 se tiene:

$$m(H_1,2) \geq 2 \cdot \frac{469}{293} \left[1 - 1 \cdot \frac{0}{5-1} \right] = 3.20$$

Para H_2:

$$m(H_2,2) \geq 2 \cdot \frac{320}{293} \left[1 - 1 \cdot \frac{1}{5-1} \right] = 1.64$$

Y para H_3:

$$m(H_3,2) \geq 1 \cdot \frac{576}{293} \left[1 - 1 \cdot \frac{4}{5-1} \right] = 0$$

Número esperado de copias	*Número de copias*	*Cadenas*
H_1 : 3.20	3	2,3,4
H_2 : 1.64	2	2,3
H_3 : 0.00	1	4

Tabla 5. Procesamiento de *schemata* después del cruce

Capítulo III

Diseño del Algoritmo

"La ciencia de la computación no trata sobre las computadoras más de lo que la astronomía trata sobre los telescopios"
Edsger Dijkstra

Para desarrollar el algoritmo se utilizó el paquete **MATLAB** (*Matrix Laboratory*), ya que permite una fácil implementación de las principales estructuras de datos de los **AG** como son, los cromosomas, fenotipos, valores de la función objetivo, así como los valores de aptitud [14]. Una población entera de cromosomas puede ser almacenada en un arreglo matricial con sólo definir el número de individuos (*NIND*) y la longitud de su representación del genotipo (*LIND*). De manera similar, las variables de diseño, o los fenotipos que se obtienen aplicando algún mapeo a los cromosomas se pueden almacenar en arreglos matriciales. El mapeo depende del esquema de decodificación utilizado. Los valores de la función objetivo y los valores de aptitud pueden ser escalares o vectoriales [13].

MATLAB es un software matemático que ofrece un entorno integrado con un lenguaje propio de programación de alto nivel, entre sus herramientas básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (*GUI*) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

MATLAB tiene una amplia variedad de funciones útiles para el desarrollo de los algoritmos genéticos, por ejemplo, la función *randperm* permite generar una permutación de números aleatorios, lo cual facilita implementar el operador de mutación, el cual se detallará más adelante. Dada su gran capacidad para manejar matrices de gran dimensión, el problema de ruteo puede ser codificado en un tiempo corto, en comparación del tiempo que tomaría codificarlo en otros lenguajes de programación como C ++ o Basic para el mismo propósito.

Estructura de datos del Algoritmo Genético

MATLAB maneja la información en forma de matrices con elementos numéricos reales o complejos. Considerando esta característica, la usaremos para diseñar el algoritmo en términos matriciales [14]. En este apartado se presentan las principales estructuras que integran el algoritmo genético, incluyendo la codificación de los cromosomas, la definición del mapa de ciudad, así como la implementación de la función objetivo y los operadores genéticos de cruce y mutación.

Codificación de cromosomas

La estructura de datos definida como Cromosomas (*Chrom*) almacena una población entera de individuos en una matriz de tamaño *NindxLind*, donde *Nind* es el número de individuos en la población y *Lind* es la longitud de la cadena que representa a esos individuos, así pues, cada renglón de la matriz corresponde al genotipo de un individuo. La representación matricial del cromosoma es la base del desarrollo del algoritmo genético.

$$Chrom = \begin{bmatrix} g_{1,1} & g_{1,2} & g_{1,3} & \cdots & g_{1,Lind} \\ g_{2,1} & g_{2,2} & g_{2,3} & \cdots & g_{2,Lind} \\ g_{3,1} & g_{3,2} & g_{3,3} & \cdots & g_{3,Lind} \\ . & . & . & \cdots & . \\ g_{Nind,1} & g_{Nind,2} & g_{Nind,3} & \cdots & g_{Nind,Lind} \end{bmatrix}$$

Para codificar a los individuos de la población se utilizó una codificación basada en la permutación de números enteros, esta consiste en codificar un *tour* por un vector cuyas componentes son las ciudades que se deben recorrer o visitar, donde el primer elemento del vector es la primera ciudad y así, sucesivamente hasta la última ciudad. El genotipo de cada cromosoma para el diseño de nuestro algoritmo se basó en este tipo de codificación, pero se dividió el cromosoma en 2 partes, la primer parte contiene la información de recorridos, es decir, el orden en que deberán ser visitados los destinos, y la segunda parte simboliza las rutas por las que se deberá llegar a dichos destinos. Por ejemplo, para un cromosoma A definido para recorrer 5 destinos se tiene:

A:	2	4	1	3	5	1	2	1	1	3
	Orden del recorrido					Rutas disponibles				

Fig.18 Codificación del cromosoma

El cromosoma anterior se interpreta así: los primeros 5 genes proporcionan el *"orden de recorrido",* esto es, el automóvil deberá ir primero al destino 2, luego al 4, al 1, al 3 y al 5 para finalmente regresar a 2. Nótese que hasta el momento el vehículo conoce el orden de recorrido, pero desconoce las rutas que conectan dichos destinos.

La segunda parte del cromosoma proporciona la información de las rutas *"disponibles"* para recorrer los destinos en el orden que se requiere, por ejemplo, para ir del destino 2 al destino 4, tenemos que la ruta disponible es la 1, y para ir del destino 4 al 1, se eligió de manera aleatoria la ruta 2 y así sucesivamente.

Implementación de la Matriz de Distancias Dinámicas

Para almacenar la información de la matriz de distancias dinámicas, **MATLAB** permite crear directamente una base de datos con un arreglo tipo célula (*cell*), en dicho arreglo cada celda de la célula puede ser una matriz o un vector que contenga la información de las rutas disponibles para un par de clientes en específico (ver la definición de matriz de distancias en el capítulo I). En la **F**igura 19 se muestra un arreglo tipo *cell* generado en **MATLAB**, observe que su orden es de 8x8 y que sus entradas son matrices de distinto orden entre sí. Por ejemplo el elemento 1x2 contiene una matriz de orden 3x8, es decir tres rutas con ocho calles, mientras que el elemento 2x1 es una matriz de orden 2x16, dos rutas con dieciséis calles.

```
Columns 1 through 4

             []    [3x8  double]    [5x11 double]    [5x15 double]
  [2x16 double]               []    [2x3  double]    [2x11 double]
  [5x15 double]    [5x13 double]               []    [4x15 double]
  [3x14 double]    [2x13 double]    [4x13 double]               []
  [5x16 double]    [2x16 double]    [7x10 double]    [4x12 double]
  [6x17 double]    [6x17 double]    [4x13 double]    [4x13 double]
  [4x17 double]    [4x17 double]    [4x13 double]    [3x13 double]
  [4x17 double]    [4x17 double]    [4x12 double]    [2x5  double]

Columns 5 through 8

  [5x15 double]    [5x15 double]    [5x15 double]    [5x15 double]
  [3x14 double]    [5x11 double]    [5x12 double]    [4x13 double]
  [5x13 double]    [4x9  double]    [5x9  double]    [4x11 double]
  [3x14 double]    [3x9  double]    [6x17 double]    [5x16 double]
             []    [6x11 double]    [8x12 double]    [7x16 double]
  [4x10 double]               []    [5x13 double]    [3x6  double]
  [4x11 double]    [3x14 double]               []    [3x14 double]
  [8x13 double]    [5x12 double]    [5x15 double]               []
```

Fig.19 Matriz de distancias dinámicas implementada en MATLAB

Operador Reparador

Considerando que la población inicial de individuos se crea de manera aleatoria, se debe tener cuidado de que ciertas restricciones que afectan el desempeño del algoritmo se cumplan. Las dos restricciones principales que debe satisfacer toda población de individuos en el algoritmo propuesto son:

1) El orden de recorridos no puede contener elementos repetidos, es decir, no se puede visitar dos veces a un mismo cliente en un solo recorrido.
2) El número de rutas disponibles no se puede superar para un par de clientes cualesquiera, esto es, se debe respetar el orden de la matriz que contiene la información de rutas.

En este trabajo se optó por reconstruir aquéllos individuos que no cumplen con dichas restricciones. La reconstrucción se llevó a cabo por medio de un nuevo operador que se denomina *"reparador"*. Este operador verifica que se satisfagan todas las restricciones del problema en todos los individuos de la población, en caso de encontrar a un individuo que no las cumpla, el operador lo "repara" y lo inserta de nuevo en la población. El código del operador reparador implementado en **MATLAB** se muestra en la **F**igura 20.

```
function NewChrom = reparador(Chrom,rutas)
% cargar tamaño de la población (Nind) y longitud del cromosoma (Lind)
[Nind,Lind] = size(Chrom) ;
for j=1:Nind,
   m=Lind/2; % número de destinos
   for i=2:Lind/2,
    m=m+1; % locus de ruta disponible
    rdisp=rutas{Chrom(j,i-1),Chrom(j,i)}; % cargar ruta disponible
    % si excede el número de rutas disponibles regreso a la primera
    if Chrom(j,m)>size(rdisp,1), Chrom(j,m)=1; end
    if i==Lind/2, % cerrar el ciclo de recorrido
        rdisp=rutas{Chrom(j,i),Chrom(j,1)};
        if Chrom(j,Lind)>size(rdisp,1), Chrom(j,Lind)=1; end
    end
    end
end

NewChrom=Chrom; % Guardar el cromosoma reparado
```

Fig. **20** Operador reparador implementado en **MATLAB**

Definición de mapa de ciudad

Se diseñó un mapa de ciudad a partir de una imagen digital y se procesó con **MATLAB**. Este mapa **no** representa una ciudad real y sólo se utiliza para ejemplificar el funcionamiento del algoritmo [Figura 21].

El mapa define de manera precisa todas las calles por donde los vehículos podrán circular. **MATLAB** tiene la capacidad de reconocer figuras conectadas (en este caso calles) a partir de imágenes digitales, asignarles un índice identificador y almacenar una gran cantidad de información en una estructura de datos que permite añadir información del entorno de la ciudad, por ejemplo, definir el sentido de circulación y la longitud de las calles, datos que son de utilidad para construir la función objetivo, la cual se describe a continuación.

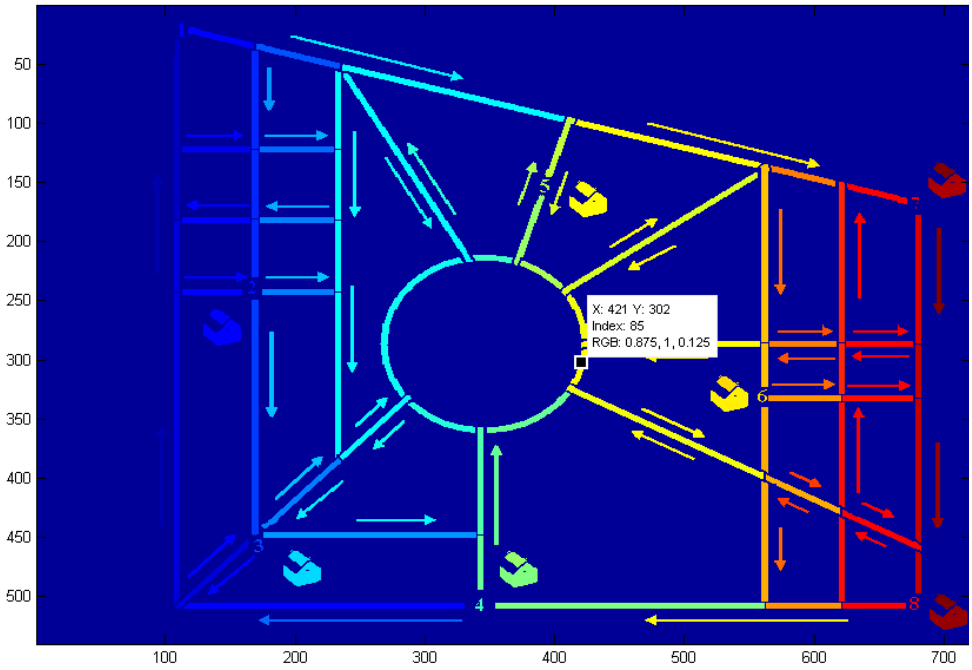


Fig.21 Mapa digital procesado con **MATLAB**

Implementación de la Función Objetivo

La *Función Objetivo* debe evaluar el desempeño de cada individuo, que en términos del problema planteado, significa optimizar (minimizar) la *distancia dinámica* para todos los caminos posibles, para ello deberá calcular la longitud de cada uno de ellos y penalizar las rutas que presenten bloqueos. Recordemos del capítulo I que la función objetivo fue definida como:

$$F.O. = \sum_{k=1}^{l} L_k + P * Nb_k(t)$$

$$L_k = \sum_{j=1}^{m} \Omega_j$$

Donde:

L_k = Longitud de la ruta k

P = Valor de penalización

$Nb_k(t)$ = Número de bloqueos en la ruta k en el instante t

Ω = Longitud de la calle j

m = número de calles por ruta

l = número de rutas alternas por cliente

La longitud Ω de cada calle se calcula con la función *Region props* de **MATLAB**, dicha función almacena varias propiedades geométricas a partir del mapa de ciudad, una de las cuales es la longitud lineal de las calles, que es la que interesa. Cuando se presente un bloqueo en una calle, se penalizará la función objetivo asignando un valor muy grande (en este trabajo se considera P=1000000), de esta manera la aptitud del *"individuo"* disminuirá, y por tanto no será seleccionado para la siguiente generación. Los bloqueos de ruta se analizan con detalle más adelante (*ver algoritmo de simulación de bloqueos*). Los valores de la función objetivo se almacenan en un vector columna de longitud *Nind* llamado "*ObjV*" [Figura 22].

```
ObjV = OBJFUN(Chrom); % Función objetivo

    ObjV = f1       individuo 1
           f2       individuo 2
           f3       individuo 3
           . . .
           fNind    individuo Nind
```

Fig. 22 Estructura de la función objetivo en MATLAB

Para aclarar las ideas anteriores se presenta el siguiente ejemplo, suponga que se tiene una matriz de población inicial de tamaño 10 generada de manera aleatoria [Figura 23].

4	3	1	2	2	3	1	1
4	2	3	1	1	2	1	4
2	4	1	3	2	1	1	3
4	3	2	1	2	1	1	3
2	4	3	1	1	2	4	3
4	1	2	3	2	1	1	1
3	2	1	4	1	2	4	3
3	1	2	4	4	2	1	1
4	1	2	3	3	1	2	1
3	1	4	2	3	1	1	2

Fig. 23 Matriz de población inicial de tamaño 10

Los renglones representan diferentes soluciones potenciales al problema, por ejemplo, considere el primer renglón solución (cromosoma 1):

4	3	1	2	2	3	1	1

Los cuatro primeros genes del cromosoma representan el recorrido que se debe seguir, es decir se inicia en la ciudad 4, luego se visita la ciudad 3, posteriormente la 1 y así sucesivamente hasta llegar nuevamente al punto de partida. La parte restante del cromosoma representa las rutas disponibles para realizar dicho recorrido.

La función objetivo recibe la información de las ruta para ir del destino *i* al destino *j* y calcula la distancia Ω para cada conjunto de calles C(*i*,*j*) que componen dichas rutas. En la siguiente tabla se muestran las distancias para el cromosoma considerado. La primera columna representa el conjunto de calles para la ruta que va del destino 4 al destino 3, la siguiente columna representa la distancia euclidiana de cada calle y su suma. Lo mismo para las siguientes columnas. Por último, en la parte inferior derecha se indica la suma total de las distancias que es el valor final que entrega la función objetivo. Del mismo modo se calculan las distancias totales del los recorridos para todos los individuos de la población y se guardan en el vector *ObjV* (**F**ig.24). Por ejemplo, el valor total de la

```
ObjV =

   5264.8
   6095.5
   6759.3
   6248.8
   5535.5
   4498.2
   7327.7
   5393.8
   4638.5
```

Fig. 24. Distancias calculadas por la función objetivo

función objetivo de la Tabla 6 (5264.8) será el primer renglón del vector *ObjV* y así sucesivamente con todos los demás individuos de la población.

C(4,3)	$\Omega_{4,3}$	*C(3,1)*	$\Omega_{3,1}$	*C(1,2)*	$\Omega_{1,2}$	*C(2,4)*	$\Omega_{2,4}$
68	98	38	334	18	119.0122	36	116
67	188	67	188	25	172	51	282.8284
63	123.6396	63	123.6396	26	122	52	154.4924
52	154.4924	60	318.2914	27	104	63	123.6396
35	173.8061	54	402.7006			69	159.6812
		48	137.4142			82	370.9605
		49	124			103	208.8284
		34	120			71	420
		12	112				
		6	124				
		5	189.4142				
$\sum=$	737.9381	$\sum=$	2173.46	$\sum=$	517.0122	$\sum=$	1836.4305
						$\sum_{k=1}^{n}\sum_{j=1}^{m}\Omega_{kj} =$	5264.8

Tabla 6. Distancia total del recorrido.

Valores de aptitud

Los valores de aptitud proporcionan una medida de la calidad de las soluciones y se obtienen por medio de la función *Ranking* implementada en el ToolBox de Algoritmos Genéticos para **MATLAB** [21]. Esta función toma como entrada al vector *ObjV* que contiene los valores de desempeño de una población con *Nind* individuos, de esta manera la función *ranking* genera un vector columna (*FitnV)* que almacena la aptitud o *"fitness"* para cada individuo de dicha población [Figura 25]. Los valores de aptitud servirán más adelante para seleccionar a los individuos que pasarán a la siguiente generación.

```
FitnV = ranking(ObjV); % Función de aptitud

   FitnV = f1       individuo 1
           f2       individuo 2
           f3       individuo 3
           . . .
           fNind    individuo Nind
```

Fig. 25 Estructura de la función de aptitud en **MATLAB**

Esta función está diseñada para problemas de minimización y utiliza por defecto un *ranking lineal* con presión selectiva igual a dos, es decir, el número máximo de copias que puede recibir un individuo se limita a dos. Por ejemplo, considere una población de 10 individuos con los siguientes valores de desempeño:

ObjV = [1; 2; 3; 4; 5; 10; 9; 8; 7; 6]

Al evaluar la función de *ranking lineal* se obtiene el vector de aptitudes *FitnV* de la Figura 26. Observe que el individuo con mejor valor de aptitud en el vector *FitnV*, recibirá 2 copias como máximo en la siguiente generación, como ya se mencionó, limitar el número de copias evita que la población sea dominada por individuos con valores de aptitud muy elevada, ya que generalmente esta situación provoca una convergencia prematura del algoritmo.

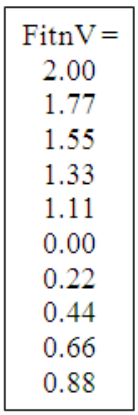


Fig. 26. Vector de aptitudes

Selección de individuos (SelCh)

La selección de los individuos más aptos se realizó usando la función de alto nivel *"select"* (disponible en el toolbox de **AG** [21]). Su estructura básica es:

SelCh = select(SEL_F, Chrom, FitnV, GGAP)

Esta función es la encargada de seleccionar a los individuos con mayor aptitud de una población (contenida en la matriz *Chrom*), y almacenar a los individuos seleccionados en una nueva población (matriz *SelCh*). Cada renglón de las matrices *Chrom* y *SelCh* corresponde a un individuo, y los argumentos de esta función corresponden a:

- *SEL_F*: es una cadena que contiene el nombre de la función de bajo nivel de selección. En este trabajo se utilizó el método de selección por rueda de ruleta "*RWS"* descrito en el capítulo anterior.
- *FitnV:* es el vector columna que contiene el valor de aptitud para cada individuo en la matriz *Chrom*. El valor de aptitud indica la probabilidad de selección esperada para cada individuo.
- *GGAP:* (*generation gap*) es un parámetro que determina el GAP generacional, si no se especifica un valor, la función asigna por defecto un GAP=1.

El parámetro GGAP es de suma importancia ya que permite especificar el porcentaje de individuos que serán reemplazados en la población original *Chrom* por los individuos más aptos de la población *SelCh* (Por ejemplo, GGAP=1 significa que se reemplaza al 100% la población *Chrom*). Si el GGAP es diferente de uno, entonces la matriz *Chrom* no será del mismo orden que la matriz *SelCh,* por esta razón se tendrá que utilizar más adelante un *operador de reinserción* para uniformar dichas poblaciones.

- *Operador de cruce (PMX)*

El operador de *cruce* de un solo punto (descrito en el capítulo II) no se puede aplicar directamente debido a que es inconsistente con las características de nuestro problema ya que algunos clientes podrían resultar repetidos o incluso no aparecer en el recorrido [1][4]. La **F**igura 27, ilustra el mecanismo de este tipo de cruce, como se puede apreciar, los clientes 4 y 5 no aparecen en el Hijo 1, mientras que los clientes 1 y 2 aparecen más de una vez. El Hijo 2 también sufre inconsistencias similares. Por tal motivo se requirió implementar un nuevo operador llamado *"cruce parcialmente apareado" (**PMX**,* por sus siglas en inglés*, partially matched crossover*).

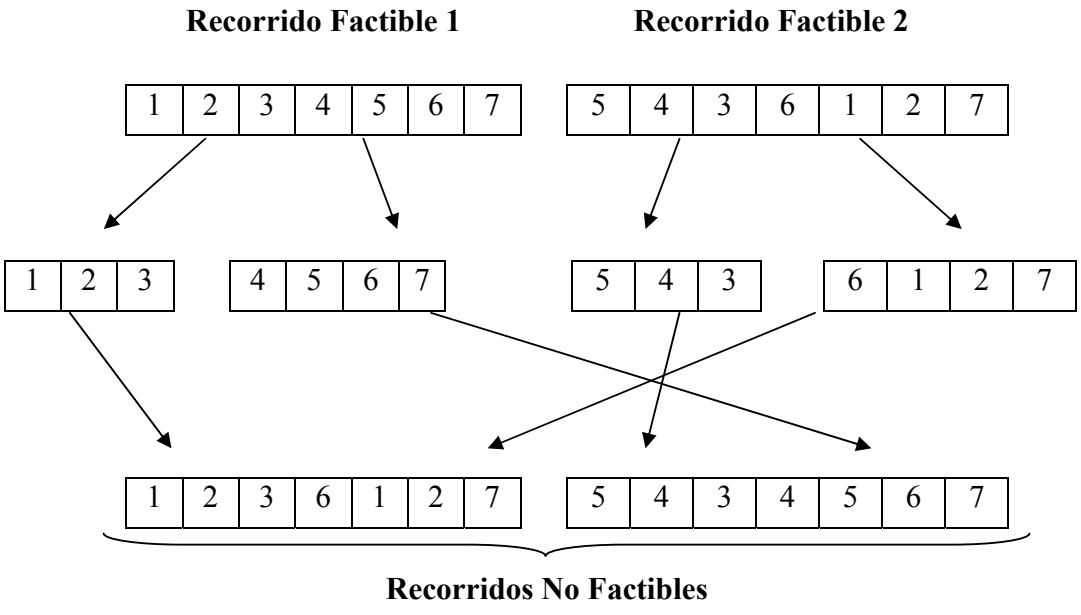


Fig.27 Cruce simple de recorridos

El operador de cruce **PMX** consiste en que dados dos cromosomas padres, el operador copia una subcadena (región de cruce) de uno de ellos directamente en las mismas posiciones del otro. Las posiciones restantes se llenan con valores que aún no hayan sido utilizados, colocándolos en el mismo orden como se encontraban en su progenitor [12].

Por ejemplo, si iniciamos con dos cadenas que juegan el rol de padres:

```
p1: [1,2,3,4,5,6,7]
p2: [3,7,6,1,5,2,4]
```

Si la subcadena seleccionada al azar de p_1 para ser insertada en p_2 es la [3,4,5], estos valores desplazarán a la subcadena [6,1,5] que ocupa las mismas posiciones en p_2. El cromosoma hijo modificado quedaría entonces así: [3,7,3,4,5,2,4]. El siguiente paso consiste en eliminar de p_2 aquellos elementos que están repetidos como consecuencia de la inserción de la subcadena, resultando [*,7,3,4,5,2,*]. El último paso, consiste en rellenar estos huecos, con los elementos de p_1 y respetando su orden. Por ejemplo, el 1 se introduce en el lugar del 3, que produce el individuo [1,7,3,4,5,2,*]; el 6 en lugar del 4. De esta manera obtenemos [1,7,3,4,5,2,6]. Similarmente se procede con el segundo cromosoma hijo (**F**ig.28). El operador de cruce **PMX** implementado en **MATLAB** se encuentra en los anexos de este trabajo.

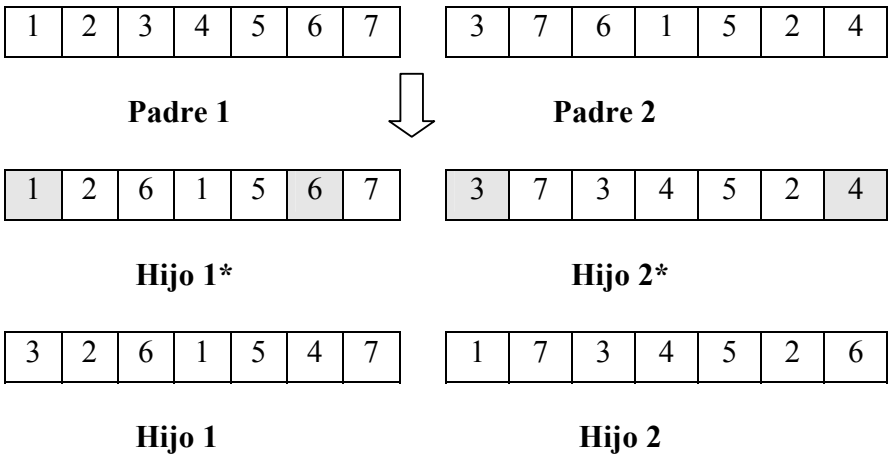


Fig.28 Descendencia después de aplicar el cruce **PMX**

Operador de Mutación

En general el operador mutación tiene una alta probabilidad de generar descendencia con recorridos no factibles, por lo que, en el contexto de nuestro problema se define la mutación como el intercambio aleatorio de genes en los cromosomas. Por ejemplo, en la **F**igura 29 los clientes 2 y 5 se intercambian con una operación de inversión.

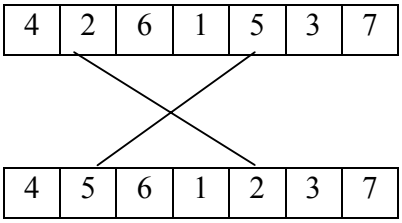


Fig. 29 Mutación

El operador se dividió en 2 partes, la primera muta el orden de los recorridos, y la segunda muta las rutas disponibles. En la Figura 30 se muestra la implementación del operador mutación en **MATLAB**. Los parámetros de entrada del operador son la población *Chrom* con su respectiva probabilidad de mutación P_m. Se debe tener cuidado al aplicar este tipo de operador ya que debe cumplir con las mismas restricciones que el operador reparador, por lo que después de ejecutar la transformación es necesario aplicar nuevamente el operador reparador para garantizar que el algoritmo arroje resultados factibles.

```
function NewChrom = mutacion(Chrom,Pm)
 [Nind, Lind] = size(Chrom) ;

 for i=1:Nind,
     if rand(1)<=Pm,
        % mutar destinos
        buffer=randperm(Lind/2); % Genera números aleatorios
        k=buffer(1); % Selecciona un numero aleatorio
        l=buffer(2); % Selecciona un numero aleatorio
        aux=Chrom(i,k);
        Chrom(i,k)=Chrom(i,l); % Cambia de lugar el gen
        Chrom(i,l)=aux;

        %mutar rutas
         buffer=randperm(Lind);
         for j=(Lind/2)+1:Lind,
              Chrom(i,j)=buffer(j-(Lind/2));
         end
      end
    NewChrom=Chrom; % Actualiza los cromosomas
 end
```

Fig. 30 Operador Mutación implementado en **MATLAB**

- *Operador Reinserción*

Este operador reinserta la descendencia en la población actual con lo que se conserva un porcentaje de individuos determinado por el parámetro GGAP. Después de aplicar este operador la población será nuevamente de orden *NindXLind*.

- *Diversidad genética*

Dado que un **AG** puede converger prematuramente cuando los individuos de la población son idénticos o muy parecidos entre sí. La búsqueda se puede estancar en un óptimo local. Para solucionar este comportamiento, existen varias técnicas que tratan de preservar la diversidad entre los individuos de la población y así, evitar una convergencia prematura. En la literatura, se han utilizado *"poblaciones múltiples"* para incrementar la calidad de las soluciones en comparación a un **AG** tradicional de una sola población.

Esta técnica consiste en evolucionar "*subpoblaciones"* por medio de un **AG** tradicional y migrar individuos de una subpoblación a otra con cierta frecuencia. La selección de los individuos a inmigrar puede ser proporcional al desempeño o uniforme. El número de individuos emigrados y la topología de la migración determinan la diversidad genética que puede ocurrir.

NOTA: Todas las subpoblaciones deben tener el mismo número de individuos.

Las topologías básicas del modelo de *poblaciones múltiples* son:

- **En Anillo**. Los individuos son transferidos de manera unidireccional entre subpoblaciones adyacentes. Por ejemplo, en la Figura 31 los individuos de la subpoblación 6 pueden migrar solamente a la subpoblación1, y los individuos de la subpoblación 1 sólo se desplazan hacia la subpoblación 2.

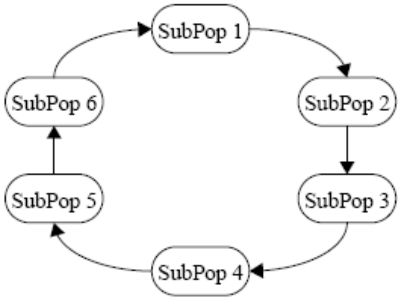


Fig. 31 Topología en Anillo

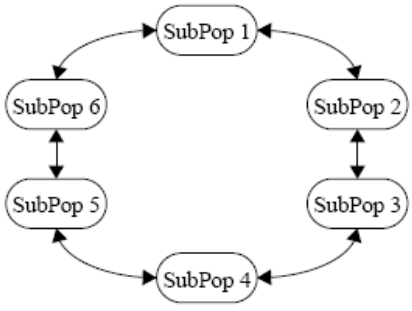


Fig. 32 Topología de vecinos

- **De Vecinos**. Al igual que la topología en Anillo, la migración se realiza entre subpoblaciones cercanas, sin embargo puede ocurrir en ambas direcciones.

- **Red completa.** Es la estrategia más general de migración. Los individuos pueden emigrar desde cualquier subpoblación a otra.

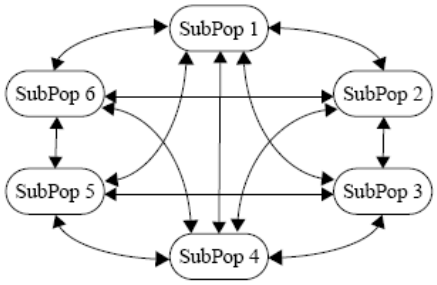


Fig. 33 Topología red completa

El *AG Toolbox* para **MATLAB** soporta subpoblaciones almacenadas en bloques contiguos dentro de un arreglo matricial [21]. Por ejemplo, en la Figura 34 se muestra una estructura de datos *Chrom*, compuesta de "*SubPop*" subpoblaciones con "*N*" individuos "*Ind*" cada una.

$$Chrom = \begin{bmatrix} Ind_1\,SubPop_1 \\ Ind_2\,SubPop_1 \\ \dots \\ Ind_N\,SubPop_1 \\ Ind_1\,SubPop_2 \\ Ind_2\,SubPop_2 \\ \dots \\ Ind_N\,SubPop_2 \\ \dots \\ Ind_1\,SubPop_{SUBPOP} \\ Ind_2\,SubPop_{SUBPOP} \\ \dots \\ Ind_N\,SubPop_{SUBPOP} \end{bmatrix}.$$

Fig. 34 Arreglo matricial de subpoblaciones.

La transferencia de individuos entre subpoblaciones está implementada en la función *migrate* del AG toolbox para **MATLAB** [21]. A continuación se presenta su estructura básica:

```
[Chrom, ObjV] = migrate(Chrom, SUBPOP, MigOpt, ObjV)
```

Donde *SUBPOP* es el número de subpoblaciones y *MigOpt* es un vector opcional de 3 parámetros:

- MigOpt(1): Porcentaje de individuos a migrar.
- MigOpt(2): Método de selección para migración.
  0 – Uniforme
  1 – Proporcional al desempeño
- MigOpt(3): Topología de migración.
  0 – Red completa
  1 – Vecinos
  2 – Anillo

Por ejemplo, la función [Chrom,ObjV] = migrate(Chrom,SUBPOP,[0.3 1 2],ObjV]) escoge 30% de individuos de una población "*Chrom*" y los reemplaza por individuos con alto desempeño desde una subpoblación adyacente con una topología en anillo, este proceso se repite por cada subpoblación *"SUBPOP"*.

Algoritmo de simulación de bloqueos

Hasta este momento no hemos considerado ninguna restricción acerca de las rutas, pero, ¿Qué pasa si una ruta se encuentra bloqueada? (por ejemplo, por alguna contingencia o una manifestación de personas). En este apartado se analiza el comportamiento del algoritmo bajo condiciones de bloqueos simulados por condiciones de tráfico comunes, como pueden ser accidentes, desviaciones por obras de mantenimiento vial, etc. [**F**igura 35]

Fig. 35 Bloqueo de ruta por accidente

Antes de experimentar el algoritmo en una situación real, por cuestiones metodológicas debe probarse o estudiar su comportamiento en un ambiente simulado, esto se realiza con base en modelos matemáticos que explican o describen el flujo vehicular lo más cerca como acontece en la realidad.

- *Enfoque macroscópico y microscópico*

Un modelo es una abstracción que representa las características más relevantes de un objeto de estudio, en función de un propósito. El estudio del flujo de vehículos se inició esencialmente con modelos macroscópicos construidos sobre bases empíricas tomadas de la dinámica de fluidos. Desde el punto de vista *macroscópico*, el flujo vehicular se concibe como un flujo compresible casi unidimensional que puede caracterizarse a través de variables como la densidad, la velocidad promedio, etc. [19].

Otro enfoque utilizado para estudiar el flujo de vehículos es el *microscópico*, en el cuál se describen con más detalle las características de cada vehículo, como es el comportamiento de los conductores respecto a la velocidad y la aceleración. Los modelos microscópicos tratan de describir la conducta de un ser humano al volante de un automóvil cuando se le presentan diferentes estímulos y obstáculos o bloqueos típicos de una vía.

El presente trabajo se basa en un modelo *microscópico* de seguimiento de vehículos que describe el comportamiento de un conductor, cuando éste es influido por el vehículo que le precede, así como su reacción ante bloqueos inesperados.

Modelo del conductor inteligente IDM

El modelo **IDM** (*intelligent-driver model*) es un "modelo de seguimiento de vehículos" que caracteriza el tráfico en un instante dado considerando las posiciones y velocidades de todos los vehículos [20]. Este modelo establece que la decisión de cualquier conductor para acelerar o frenar depende únicamente de la distancia relativa s_α respecto al vehículo que inmediatamente tiene al frente de él y a la diferencia de sus velocidades Δv_α (**F**ig.36).

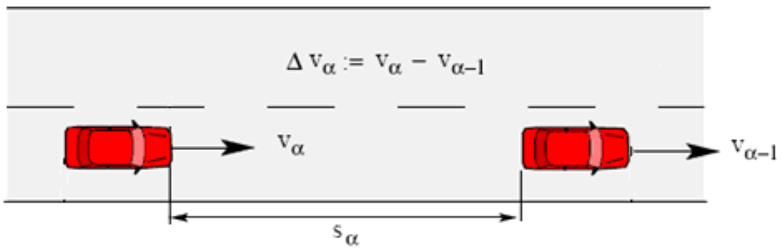


Fig. 36 Modelo IDM

La siguiente ecuación diferencial representa dicho modelo:

$$\dot{v}_\alpha IDM\left(v_\alpha, s_\alpha, \Delta v_\alpha\right) = a\left[1 - \left(\frac{v_\alpha}{v_0}\right)^\delta - \left(\frac{s^*\left(v_\alpha, \Delta v_\alpha\right)}{s_\alpha}\right)^2\right]$$

Donde:

$$s^*\left(v, \Delta v\right) = s_0 + Tv + \frac{v\Delta v}{2\sqrt{ab}}$$

v_0 = Velocidad deseada cuando se conduce por un camino libre.
T = Brecha de tiempo
a = Aceleración en un día común de tráfico.
b = Desaceleración por frenado.
s_0 =Separación de seguridad entre vehículos,
δ =Exponente de aceleración,

La aceleración instantánea $\dot{v}_{\alpha}IDM$ se compone de 2 términos, el primero expresa una aceleración libre *"deseada"* $a[1-(v_{\alpha}/v_0)^{\delta}]$, es decir, cuando no hay vehículos en el camino, y el segundo una desaceleración de frenado inducida por la presencia de algún vehículo en el camino.

Cuando el vehículo alcanza su "velocidad deseada" v_0 (la cual en este trabajo llamaremos *velocidad de crucero*), la aceleración libre se hace nula para mantener su velocidad constante.

El término de frenado se basa en una comparación entre la "distancia relativa deseada" s^*, y la distancia relativa actual del vehículo que va adelante. Si la distancia relativa se aproxima a s^*, entonces la desaceleración de frenado compensa el término de la aceleración libre, produciendo una aceleración casi nula.

Esto significa que, s^* crece dinámicamente cuando se aproxima a vehículos lentos y decrece cuando el vehículo delantero es veloz [19] [20].

- *Parámetros del modelo del conductor inteligente*

En general, cada conductor puede tener sus propios parámetros, por ejemplo, los camiones se caracterizan por valores bajos de v_0, a y b. Los conductores cautelosos poseen una brecha de tiempo grande T, mientras que los agresivos se caracterizan por un valor bajo de T con valores altos de v_0, a y b [20]. Por supuesto existen muchos factores que determinan las condiciones de tráfico que son difíciles de simular, es el caso de las condiciones del suelo; si está mojado o si está nevando; si hay una falla eléctrica que afecte el sistema de semáforos, etc.

Este modelo no considera los cambios de carril, tampoco rebases que pudieran ocurrir en un día normal de tráfico, ni mucho menos el caso de colisiones, dado que siempre se guarda intencionalmente una distancia mínima de seguridad entre autos con una brecha amplia de reacción de frenado [19]. Aquí es pertinente recordar, que un modelo matemático no es una copia exacta de una parte o sistema real, sino una "simplificación" o idealización de la realidad, el cual incluye las características esenciales de la realidad modelada. Estos aspectos son los que permiten que un modelo describa de manera adecuada un fenómeno.

De acuerdo al objetivo de este trabajo, es suficiente tomar en cuenta sólo la velocidad instantánea del flujo vehicular como parámetro para determinar los

bloqueos de rutas. Por ejemplo, una velocidad promedio cercana a cero indicará que existe un flujo extremadamente lento y puede considerarse como un bloqueo.

En la **F**igura 37 se muestra la implementación en **MATLAB** de la función de velocidad del modelo *IDM* asignando valores numéricos a los parámetros (velocidad deseada de crucero, la brecha de tiempo, separación mínima, etc.), básicamente dicha función resuelve la ecuación diferencial del modelo **IDM** para diferentes valores de distancias relativas s^*. Esta velocidad se utiliza en el algoritmo para determinar la velocidad promedio de un flujo vehicular simulado, recuerde que una calle se considera bloqueada cuando la velocidad promedio de dicho flujo es prácticamente nula.

```
function V = fv(s,delta_v)
% Luis Fernando M. Mosqueira
% 25 - 04 - 2009
% Datos de entrada
% s = distancia relativa entre vehículos
% delta_v= diferencia de velocidades entre vehículos
% Parámetros:
v0=15.0;    % Velocidad  de crucero
delta=4.0; % Factor de aceleración
a=0.5;      % Aceleración 0,5
b=3.0;      % Desaceleración
T=3;        % Tiempo de reacción al frenado 2,5
s0= 20;     % distancia mínima entre autos
dt=0.5;     % Brecha de tiempo 0.5 s
kmax=200;  % Número de iteraciones para la aproximación
Vel=0;      % Inicializar velocidad
% Aproximación de la velocidad instantánea
        for k=0:kmax,

        s_star = s0 + Vel*T+(Vel*delta_v)/(2*sqrt(a*b));
        acc = a * (1 - (Vel/v0)^delta - (s_star/s)^2);
        Vel=Vel+acc*dt;
        if Vel<0.0, Vel=0.0; end

        end
 V=Vel; % Regresa el valor de la velocidad
```

Fig. 37 Función implementada en **MATLAB** para calcular la velocidad promedio del un flujo vehicular con el modelo **IDM**

Capítulo IV
Experimentación y Resultados

"La vida es el arte de sacar conclusiones suficientes a partir de datos insuficientes"
Samuel Butler

El propósito de este apartado es probar el algoritmo sometiéndolo a diferentes escenarios, primero se analiza el comportamiento del algoritmo en un ambiente libre de bloqueos, luego se bloquean intencionalmente ciertas rutas del recorrido para observar su efecto en el algoritmo y finalmente se simula un flujo vehicular para generar un congestionamiento en las rutas y de esta manera generar bloqueos aleatorios. Para la parte genética del algoritmo se crearon dos poblaciones iniciales aleatorias de individuos, y se midió el desempeño de ambas utilizando el método de *ranking lineal* con presión selectiva igual a dos. La selección de los individuos con mayor aptitud se realizó con el método de *rueda de ruleta* **RWS**, y se aplicaron los operadores de *mutación* y *cruce PMX* con diferentes valores de probabilidad. Aunque los operadores genéticos fueron diseñados especialmente para resolver el problema, éstos pueden llegar a generar soluciones no factibles, por lo que se implementó además un *operador reparador* para garantizar que se cumplan todas las restricciones del problema. La *reinserción* de la población se llevo a cabo en base al desempeño, es decir se reemplazaron a los individuos menos aptos en cada generación, sin embargo se conservó un porcentaje de la población menos apta para aumentar la diversidad de las soluciones y evitar óptimos locales. Dicho porcentaje se determinó ajustando el parámetro **GGAP** del algoritmo. Finalmente, cada cierta generación se "*emigraron*" individuos entre las poblaciones con el fin de balancear la exploración y la intensificación del espacio de búsqueda, es decir para encontrar muchas soluciones con un buen valor de desempeño. Para la parte de simulación de bloqueos, se comenzó con un ambiente libre de obstáculos para observar la rapidez de convergencia hacia la solución óptima. Después se analizó de forma dinámica en presencia de rutas bloqueadas por medio de la simulación de condiciones de tráfico, tomando como base el modelo del conductor inteligente **IDM** descrito en el capítulo anterior**.** Debido a que el algoritmo híbrido se compone de una parte genética, su desempeño es medido por el grado de evolución que alcanza durante los experimentos. No obstante, debido a la naturaleza estocástica de los **AG**s, es necesario evaluar el resultado promedio de varios experimentos, ya que difícilmente se repetirán los resultados de un experimento a otro.

Por tal motivo se calculó la curva promedio de los mejores cromosomas y su desviación estándar para medir el desempeño del algoritmo en cada evaluación. Los resultados de los experimentos permitieron "*ajustar"* o *"calibrar"* los parámetros del algoritmo.

El algoritmo fue implementado en el lenguaje de programación **MATLAB 7.8 (R2009a)** y los experimentos fueron realizados en una máquina con procesador Intel Centrino Duo a 1.60 Ghz y 1,23 Gb de memoria RAM.

Interfaz gráfica del algoritmo

Para realizar los experimentos y poder manipular los parámetros del algoritmo con facilidad, se diseñó una interfaz gráfica de usuario (**GUI**, del inglés *Graphic User Interface*). La siguiente figura muestra la interfaz básica del algoritmo. La codificación del mapa utilizado se puede consultar en los Anexos de este trabajo (ver sección A.4).

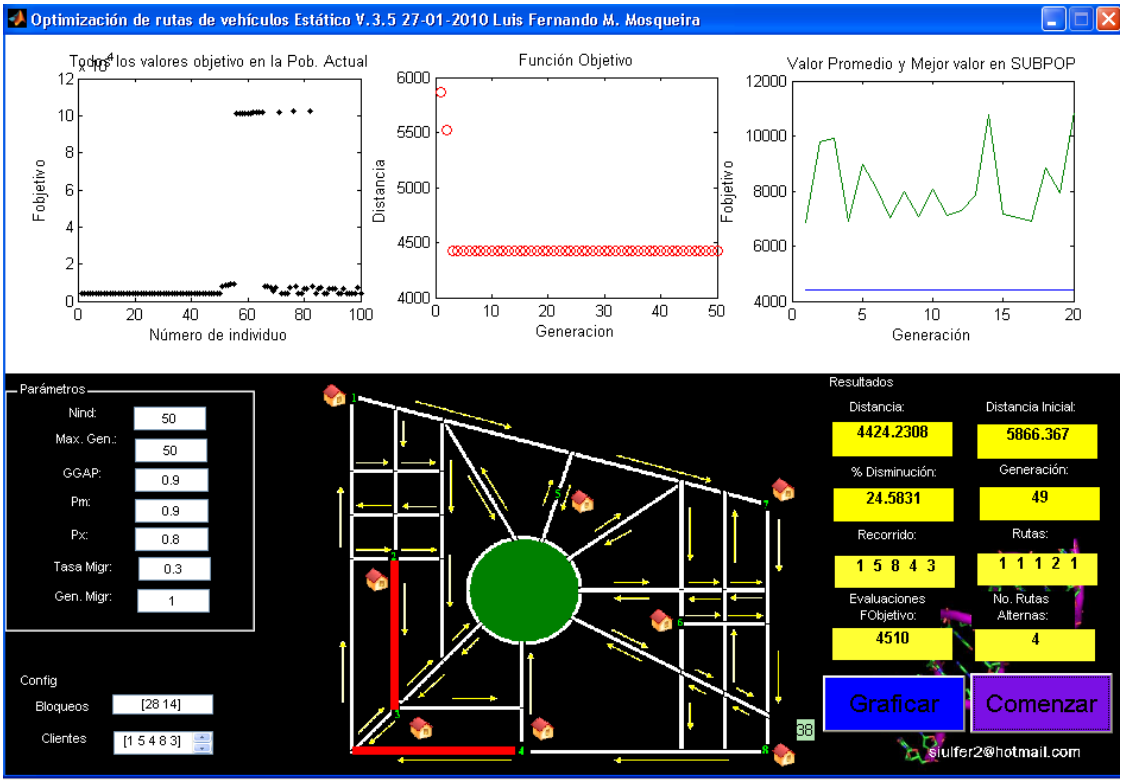


Fig. 38 Interfaz Gráfica del algoritmo híbrido

La interfaz gráfica de la Figura 38 se compone de los siguientes elementos:

- **Panel de parámetros**: Permite controlar los principales parámetros del algoritmo (Figura 35), por ejemplo el número de individuos en la población (*Nind*), el número máximo de generaciones para evolucionar a la población (*Max.Gen*), el GAP generacional (*GGAP*), la probabilidad de mutación y cruce (*Pm* y *Px*), el porcentaje de individuos a migrar entre las poblaciones (*Tasa. Migr*) y la generación en la que dicha migración se realiza (*Gen. Migr*).

Fig. 39 Panel de parámetros del algoritmo

- **Panel de configuración:** Este panel se definen los clientes a visitar, además permite bloquear manualmente las calles de las rutas con fines experimentales. Por ejemplo, en la Fig. 40 se tienen bloqueadas las calles denotadas por los números 28 y 14. Y se deberán visitar a los clientes 1, 5 4, 8 y 3.

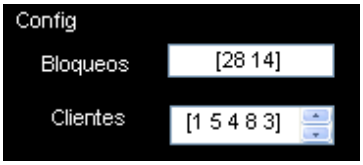


Fig. 40 Panel de configuración

- **Panel de resultados:** Este panel muestra los resultados obtenidos por el algoritmo (Figura 41). Por ejemplo, el valor de la distancia de la ruta, el % de disminución con respecto a la distancia inicial, el recorrido óptimo y las mejores rutas para realizar dicho recorrido. Incluso se muestran el número de evaluaciones de la función objetivo y el número de rutas alternas encontradas.

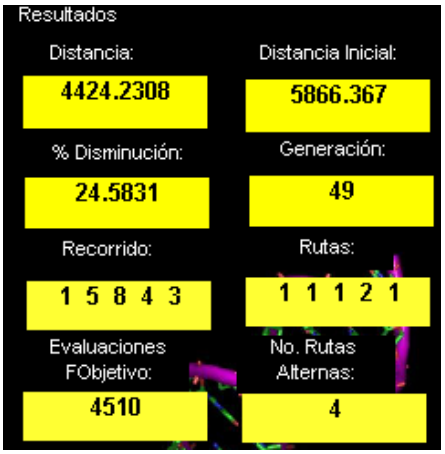


Fig. 41 Panel de resultados

- **Botón Comenzar:** Este botón el algoritmo comienza resolver el problema con los parámetros de entrada configurados en los paneles antes descritos.

- **Botón Graficar:** Grafica la ruta óptima en el mapa de ciudad por la cual el vehículo repartidor deberá visitar a todos sus clientes.

- **Gráfica de la población:** Esta gráfica muestra el valor de la función objetivo de todos los individuos de la población (Figura 42).

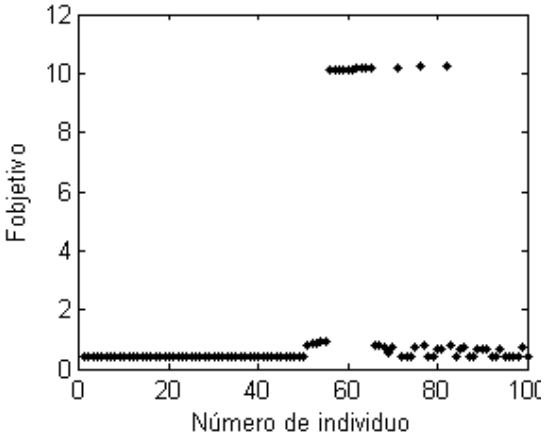


Fig. 42 Gráfica de la población

- **Gráfica de la función objetivo:** Esta gráfica muestra el valor de la función objetivo contra el número de generaciones. Esta gráfica permite visualizar la convergencia del algoritmo.

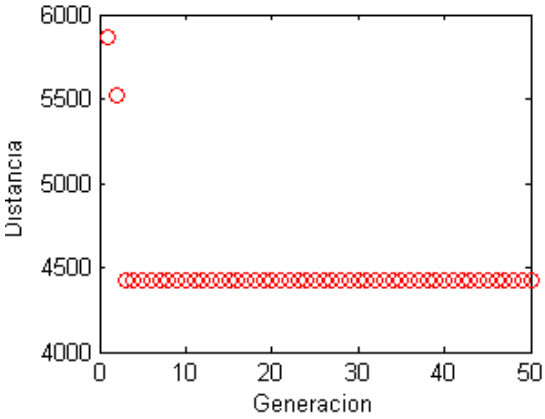


Fig. 43 Gráfica de la función objetivo

- **Gráfica promedio y mejor valor:** Esta gráfica muestra el valor promedio de la función objetivo de todos los individuos de la población (línea verde) y el mejor valor encontrado por el algoritmo por cada generación (línea azul).

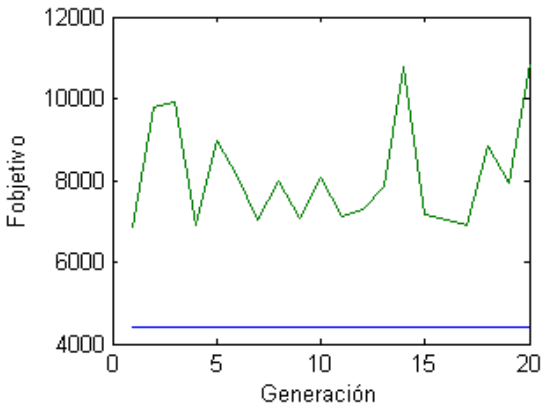


Fig. 44 Gráfica de la función objetivo

- **Mapa de ciudad:** Este mapa permite visualizar todas las calles de las rutas por donde pueden circular los vehículos, así como las calles que se encuentran bloqueadas. Por ejemplo, en la Figura 45, los rectángulos rojos representan calles obstruidas. Además en este mapa es donde se muestra la ruta óptima cuando se presiona el botón "*graficar*".

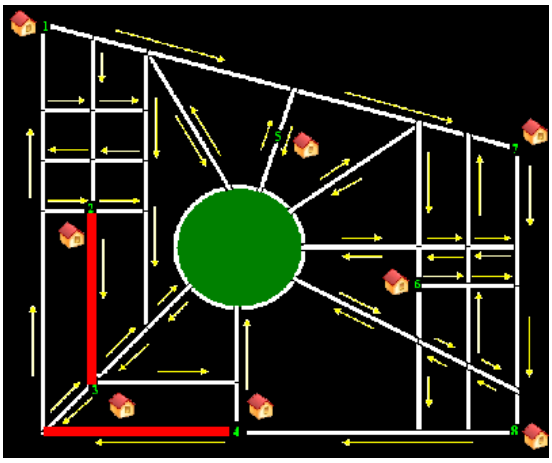


Fig. 45 Mapa de ciudad

- **Gráfica de soluciones:** La interfaz también permite obtener la gráfica de los mejores individuos de la población, es decir, las mejores rutas para realizar el recorrido. En la Figura 46 se muestran 4 soluciones encontradas por el algoritmo en donde cada línea de la gráfica representa a un individuo, los valores del *eje x* representan las posiciones ("*locus*") de los *genes* en el *cromosoma* del individuo, y los valores del *eje y*, los valores asignados a dichos genes, es decir los *alelos*. Por ejemplo, la línea roja de la gráfica representa al individuo A codificado por el cromosoma:

A:	1	5	8	4	3	1	1	1	2	1
	Orden del recorrido					Rutas disponibles				

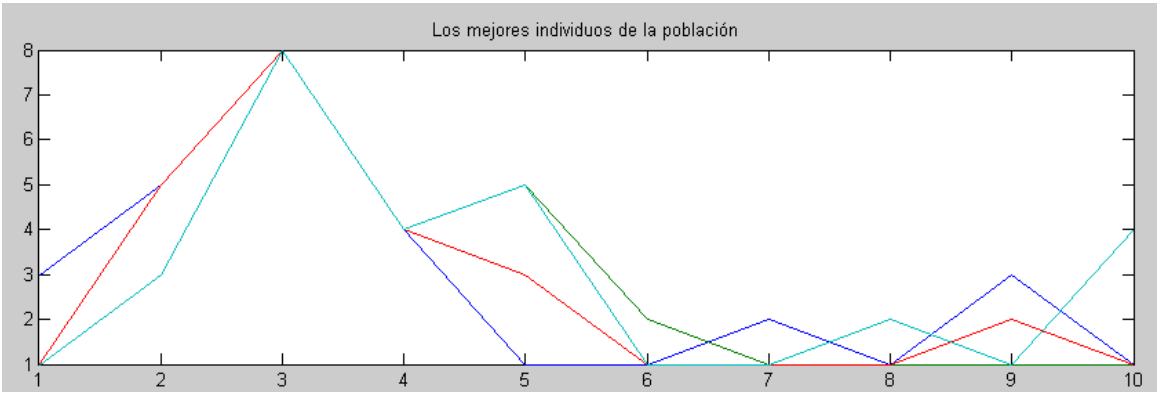


Fig. 46 Gráfica de soluciones.

Criterios para la experimentación

La calibración de los parámetros es desde luego una tarea compleja y requiere de mucho tiempo de procesamiento, esto se debe a que por cada variación de un parámetro es necesario ejecutar varias veces el algoritmo para obtener un muestreo de su comportamiento y así poder estimar su eficiencia. En este trabajo los experimentos se realizaron variando los siguientes parámetros del algoritmo:

- GAP Generacional (*GGAP*)
- Probabilidad de mutación (*Pm*)
- Probabilidad de cruce PMX (*Px*)
- Número de destinos o ciudades a visitar (*NVAR*)

Los ensayos se realizaron con un ciclo máximo de 50 generaciones con 50 individuos por población, como el algoritmo contiene dos poblaciones (una para explorar y una para intensificar el espacio de búsqueda), el número total de individuos evaluados por el algoritmo fue de 100 (*NIND*=100). La tasa de migración de individuos entre dichas poblaciones se fijó en 0.6 con base en la observación del efecto que presentó con el operador de mutación sobre la función objetivo (Figura 47). Observe que para valores de la tasa de migración mayores a 0.6 con una probabilidad alta de mutación la función objetivo tiende a aumentar su valor. Los valores asignados a cada parámetro se muestran en la Tabla 9. El número de combinaciones "*Nc*" que se pueden formar con estos valores de parámetros esta dado por la siguiente expresión:

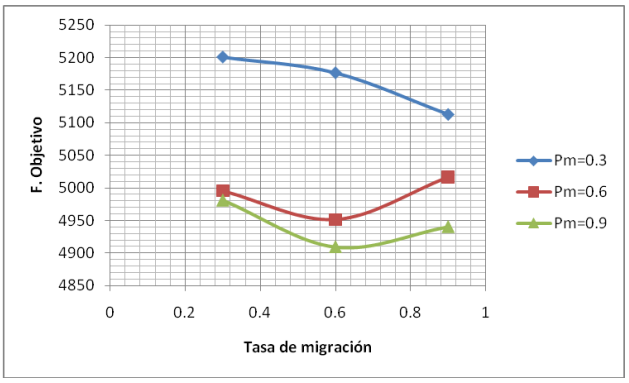


Fig. 47 Tasa de migración vs Función objetivo

Parámetro	Valores
Px	0,0.25, 0.5, 0.75, 1
Pm	0,0.25, 0.5, 0.75, 1
GGAP	0.3, 0.6, 0.9

Tabla. 9 Variación de parámetros del algoritmo

$$Nc = nPx \cdot nPm \cdot nGGAP$$

Donde *nPx* representa el número de valores asignados para el operador de cruce *Px*, *nPm* el número de valores para la mutación *Pm*, y *nGGAP* el número de valores para el *GAP* generacional, por lo tanto se tienen: $Nc = 5 \cdot 5 \cdot 3 = 75$ combinaciones. Cada experimento se realizó 100 veces por cada variación de parámetro. Por lo tanto, el número total de ensayos efectuados por cada experimento fue de *7500!*. Lo que se consideró como una muestra representativa para estimar la eficiencia del algoritmo.

Ambiente en MATLAB sin bloqueos para un recorrido de 4 clientes

Uno de los aspectos más complicados para lograr una implementación exitosa de los algoritmos genéticos es la calibración de los parámetros [10]. Los niveles para las probabilidades de mutación y cruce, así como el *GGAP* se variaron experimentalmente y se observaron los efectos que éstos tienen sobre los valores de la función objetivo. En la siguiente tabla se muestran los resultados obtenidos. La columna 4 representa el promedio de las mejores soluciones obtenidas, y la columna 5 muestra sus desviaciones estándar. La columna de distancias iniciales indica los valores iniciales encontrados por el algoritmo y en la siguiente columna se proporciona el porcentaje de disminución obtenido al encontrar el óptimo. Por último, se muestra la generación promedio en la cual el algoritmo encontró la mejor solución.

Px	*Pm*	*GGAP*	$\overline{Mejor}$	*Desv.Estándar*	*Máx.*	*Mín*	$\overline{Dist.Inicial}$	*%Dism.*	$\overline{Generación}$
0.1	0.1	0.6	3972.92	3.50E+01	4011	3941	4111	3.26	21
		0.9	3953.30	2.65E+01	4011	3941	4163	4.92	17
	0.5	0.6	3941.39	3.66E-12	3941	3941	4146	4.81	15
		0.9	3941.39	3.66E-12	3941	3941	4140	4.67	12
	0.9	0.6	3941.39	3.66E-12	3941	3941	4163	5.17	10
		0.9	3941.39	3.66E-12	3941	3941	4131	4.49	4
0.5	0.1	0.6	3949.80	2.29E+01	4011	3941	4135	4.36	13
		0.9	3941.39	3.66E-12	3941	3941	4128	4.39	11
	0.5	0.6	3941.39	3.66E-12	3941	3941	4158	5.10	14
		0.9	3941.39	3.66E-12	3941	3941	4124	4.31	6
	0.9	0.6	3941.39	3.66E-12	3941	3941	4135	4.58	8
		0.9	3941.39	3.66E-12	3941	3941	4135	4.59	4
0.9	0.1	0.6	3944.19	1.38E+01	4011	3941	4137	4.53	14
		0.9	3941.39	3.66E-12	3941	3941	4132	4.41	6
	0.5	0.6	3941.39	3.66E-12	3941	3941	4116	4.13	9
		0.9	3941.39	3.66E-12	3941	3941	4123	4.30	4
	0.9	0.6	3941.39	3.66E-12	3941	3941	4117	4.14	6
		0.9	3941.39	3.66E-12	3941	3941	4163	5.19	3

Tabla 7. Parámetros para el experimento sin bloqueos para 4 clientes (*NVAR*=4)

En las siguientes gráficas se muestra el comportamiento de los parámetros estudiados en la tabla 10 para un recorrido de 4 clientes. Como se puede advertir en las figuras 48 y 49, la mutación tiene un efecto positivo sobre los valores de la función objetivo, los cuales convergen rápidamente a la solución óptima (aprox. 3940 [m].) para probabilidades de mutación mayores a 0.5.

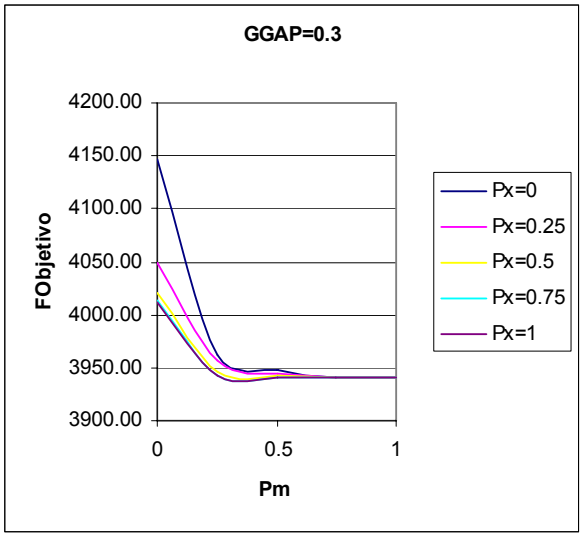


Fig. 48 Efectos de la mutación y cruce sobre la función objetivo GGAP=0.3

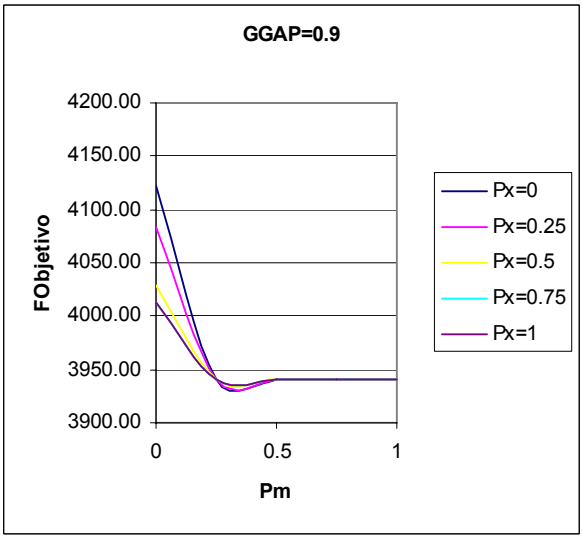


Fig. 49 Efectos de la mutación y cruce sobre la función objetivo GGAP=0.9

En las figuras 50 y 51 se advierte que con GGAP=0.9 la dispersión de las soluciones es prácticamente nula en el intervalo de $Pm \geq 0.5$ y los valores de Px no parecen tener un efecto significativo en la dispersión dentro de dicho intervalo. Por otro lado con GGAP =0.3 los datos aumentan considerable su dispersión.

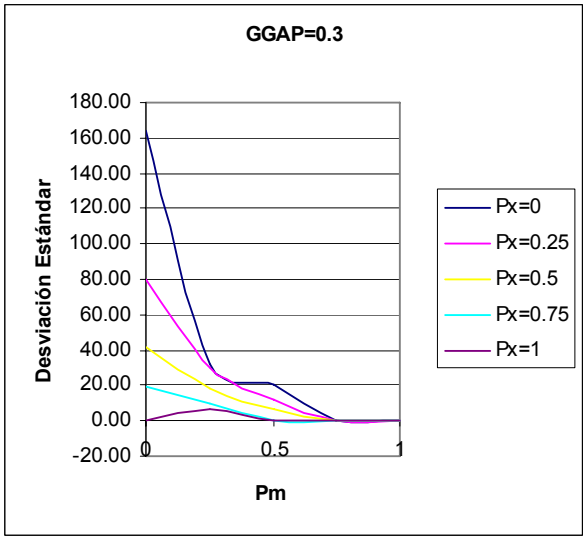


Fig. 50 Efectos de la mutación y cruce sobre la desviación estándar GGAP=0.3

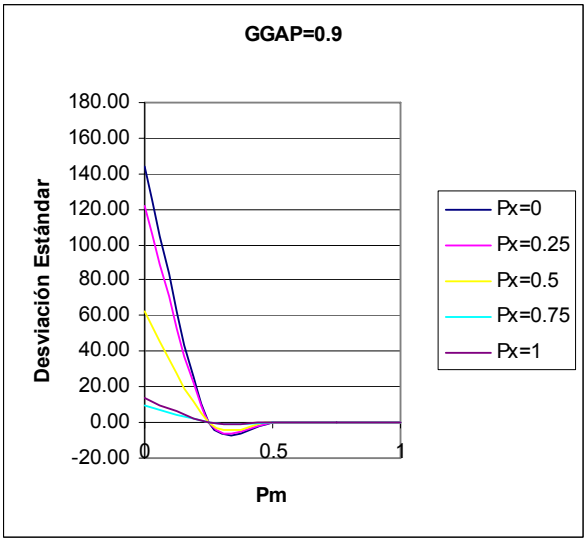


Fig. 51 Efectos de la mutación y cruce sobre la desviación estándar GGAP=0.9

Sin embargo, con base en las figuras 52 y 53 se percibe como el operador de cruce sí tiene un efecto favorable sobre la convergencia del algoritmo para valores entre $0.5 \leq Px \leq 0.75$, intervalo en el cual, el algoritmo converge aproximadamente en la generación número 5. Sin embargo, para los casos extremos (Px=0 y Px=1) el operador de cruce parece provocar un retardo en la convergencia de la solución.

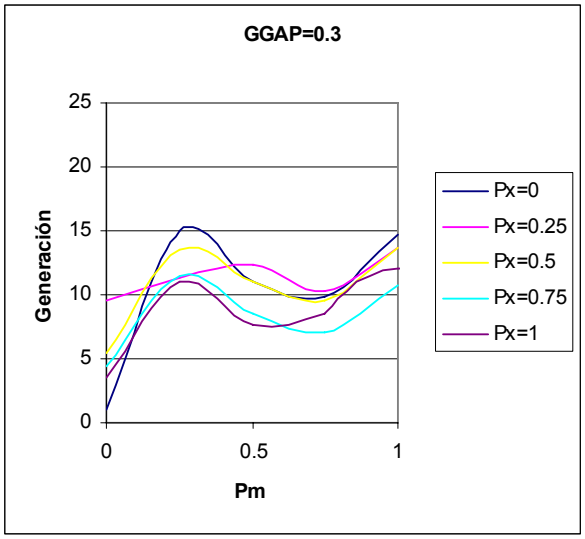


Fig. 52 Efectos de la mutación y cruce sobre el número de generaciones GGAP=0.3

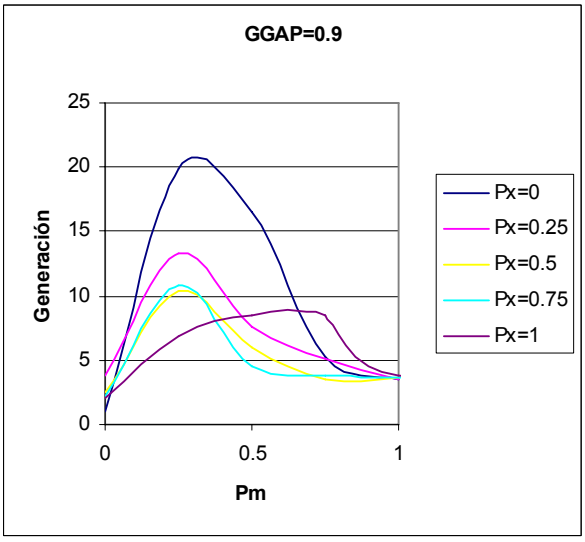


Fig. 53 Efectos de la mutación y cruce sobre el número de generaciones GGAP=0.9

- Apoyado en los experimentos descritos, los parámetros elegidos para un recorrido de cuatro clientes fueron:

- ☑ Probabilidad de mutación: $Pm \geq 0.5$
- ☑ Probabilidad de cruce: $0.5 \leq Px \leq 0.75$
- ☑ GGAP=0.9.

Las siguientes figuras (Fig.54 y Fig.55) muestran el valor de la función objetivo vs el número de generaciones para diferentes valores de probabilidad de los operadores de cruce y mutación. En la figura 54 se pone de manifiesto la rápida convergencia del algoritmo cuando se asignan valores en el intervalo $0.5 \leq Px \leq 0.75$ y Pm=0.5. Note como el algoritmo converge prácticamente en la primera generación a la solución óptima.

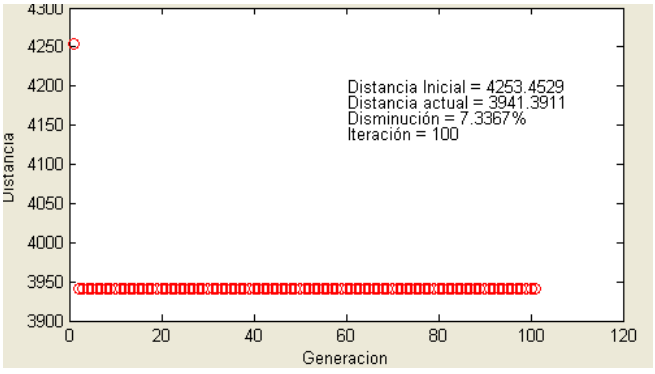


Fig. 54 Gráfica de la Función Objetivo vs Generaciones con Pm=0.5, Px=0.7

En la figura 55 se percibe un incremento en el número de generaciones para un valor de cruce muy alto (Px=1), sin embargo la solución encontrada sigue siendo óptima debido a la baja dispersión de las soluciones. El valor óptimo de la función objetivo para este experimento resultó ser en promedio de **μ = 3963.86** [m] con una desviación estándar **σ = 15.55** [m]. El mejor individuo de la población resultó ser el cromosoma:

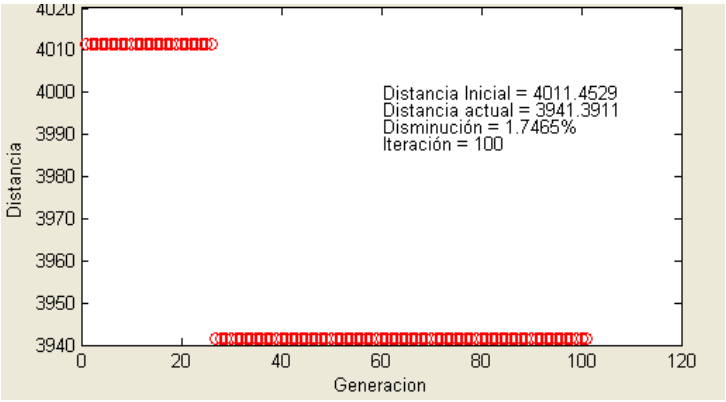


Fig. 55 Gráfica de la Función Objetivo vs Generaciones con *Pm*=0.5, *Px*=1 y *GGAP=0.9*.

A:	4	1	2	3	1	1	1	1

Recuerde que la primera parte de la codificación cromosómica indica el orden del recorrido (4-1-2-3-4) y la segunda parte las rutas disponibles para realizar dicho recorrido (1-1-1-1). La **T**abla 8 detalla el recorrido y muestra las rutas disponibles incluyendo todas las calles por ruta. Por ejemplo, el primer renglón de la tabla indica que se debe partir desde el cliente 4 para luego visitar al cliente 1 por la ruta 1, la cual está constituida por las calles denotadas por los dígitos 14, 8, 7, 6 y 5.

Recorrido	**Ruta disponible**	**Calles de la ruta**
4 – 1	1	14 8 7 6 5
1 – 2	1	18 25 26 27
2 – 3	1	28
3 – 4	1	35 52 63 69 82 103 71

Tabla 8 Orden del recorrido con calles de ruta.

Para ilustrar lo anterior, observe en la figura 56 el mapa de ciudad con la ruta óptima marcada en azul, los recuadros indican las calles por donde se debe realizar el recorrido.

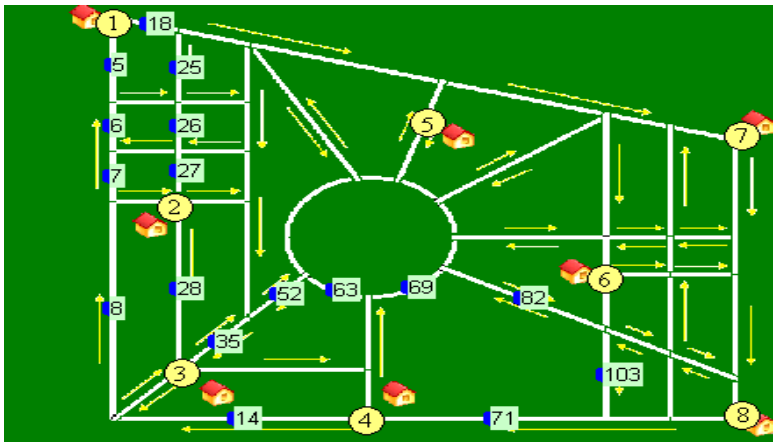


Fig. 56 Mapa de ciudad donde se muestra la ruta óptima para realizar el recorrido.

Ambiente en MATLAB si se consideran bloqueos y condiciones de tráfico

En esta sección se aborda la cuestión simulando condiciones de tráfico con bloqueo en rutas con el propósito de poner a prueba el algoritmo desarrollado en este trabajo. A diferencia de la sección anterior, los experimentos virtuales realizados fueron de carácter dinámico ya que las rutas analizadas por el algoritmo dependen del tiempo y de situaciones imprevistas. Las pruebas se basaron en la penalización de la función objetivo. Esta idea general consistió en añadir a la función objetivo una cantidad (la penalización) que guarda relación con la violación de restricciones que los individuos cometen, por ejemplo, cuando circula un vehículo por una calle bloqueada; Dicha cantidad simplemente cuenta el número de restricciones violadas y el denominado costo esperado de reconstrucción, es decir, el coste asociado a la conversión de dicho individuo en otro que no viole ninguna restricción.

Se asumió que una ruta está bloqueada cuando la velocidad promedio del flujo vehicular es muy baja o cercana a cero. La velocidad de un automóvil en un determinado instante, depende de la distancia relativa del vehículo que tiene por delante denotada por "*s**". En el modelo **IDM** se puede fijar un parámetro de distancia de seguridad, que representa la separación mínima entre los vehículos para evitar una colisión considerando el tiempo de reacción de frenado y las condiciones de aceleración del conductor. La **F**ig.57 muestra la velocidad de un vehículo en función del tiempo con diferentes valores de distancia relativa. Nótese como la velocidad aumenta conforme el auto que tiene por delante se aleja (valores de "s*" grandes), y por el contrario, la velocidad decrece hasta cero cuando se alcanza la distancia mínima de seguridad de separación, en este caso de 2m.

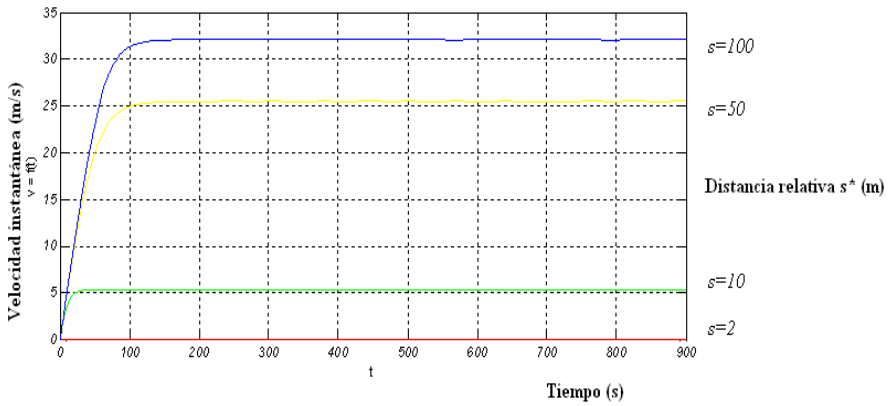


Fig. 57 Gráfica de velocidad (m/s) instantánea vs. el tiempo(s) y la distancia relativa "s" (m)

La siguiente figura ilustra como varía la velocidad conforme la distancia relativa s* aumenta.

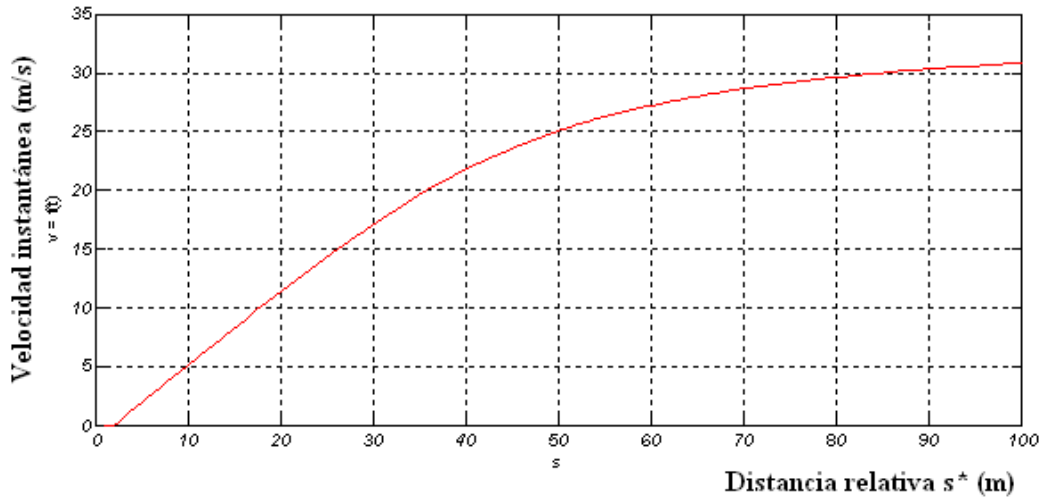


Fig. 58 Gráfica de velocidad instantánea vs. la distancia relativa entre autos.

El algoritmo de simulación genera un flujo vehicular, y asigna a cada automóvil parámetros aleatorios, por ejemplo su velocidad, su distancia relativa respecto a los demás, su posición, etc. Todos estos valores se almacenan en una matriz de orden *Nautos* x *NParámetros,* donde *Nautos* se refiere al número de autos presentes en el instante *t*, y *NParámetros* indica los valores necesarios para modelar las condiciones de tráfico. Por otro lado, cada vehículo tiene asociado un destino, el cual se elige de manera aleatoria y con una probabilidad asociada, es decir, hay destinos que tienen más peso para ser elegidos con el fin de simular una condición de bloqueo.

Al correr la simulación, el flujo vehicular puede volverse lento conforme las distancias relativas disminuyen o por el contrario, puede darse el caso de que en cierto instante los autos circulen de manera fluida dependiendo de factores aleatorios. En ambos casos el algoritmo híbrido propuesto debe encontrar la ruta óptima para ese instante.

Recorrido de 8 clientes con 3 rutas bloqueadas

En este experimento se mantuvieron intencionalmente 3 rutas bloqueadas para observar el comportamiento del algoritmo. La siguiente figura muestra con rectángulos rojos dichas rutas con bloqueo, cuyos índices de identificación en el mapa son: 28, 74 y 83.

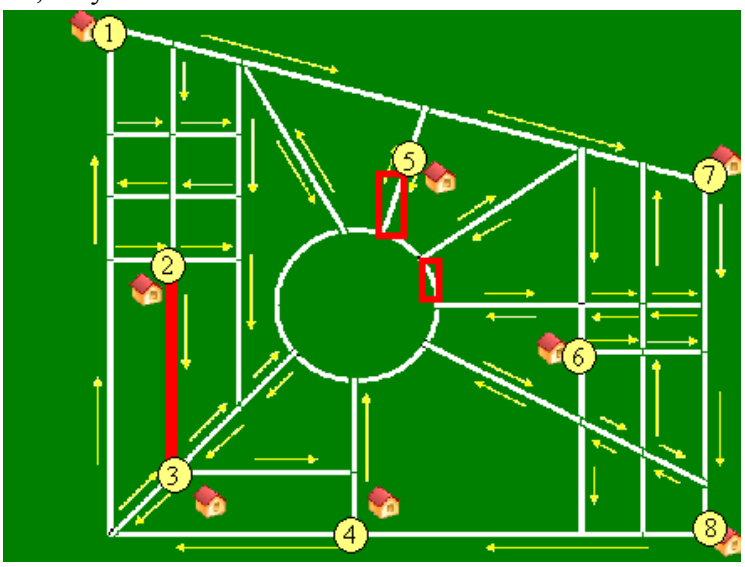


Fig.59 Rutas con bloqueo

Al igual que en los experimentos estáticos, se realizaron 7600 ensayos para calcular la curva promedio de las mejores soluciones. En las Figuras 60 y 61 se puede percibir que para un valor de mutación mayor o igual a 0.5, la función objetivo converge a un valor aproximado de 7500. Note que el parámetro GGAP no parece tener un efecto importante en la función objetivo.

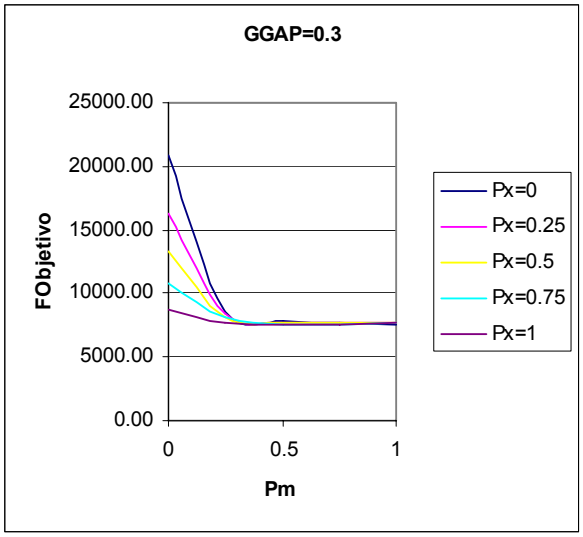


Fig. 60 Efectos de la mutación y cruce sobre la función objetivo GGAP=0.3

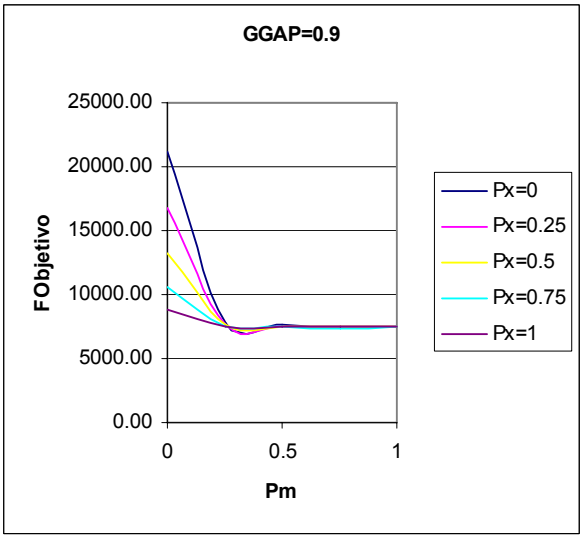


Fig. 61 Efectos de la mutación y cruce sobre la función objetivo GGAP=0.9

En la Fig. 63 se percibe que para valores grandes de probabilidad de cruce cercanos a *Px*=1, el número de generaciones crece considerablemente (generación 65). Sin embargo, para valores de *Px*= 0.75 la convergencia de la solución mejora y se manifiesta aproximadamente en la generación 30 con *Pm*= 0.5. Observe como el parámetro GGAP tiene un efecto positivo sobre el número de generaciones, ya que la convergencia disminuye aproximadamente de la generación 50 (GGAP=0.3) a la generación 30 con GGAP=0.9.

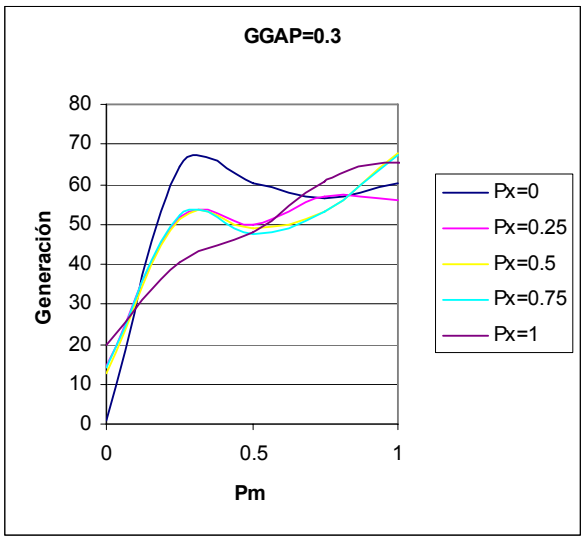


Fig. 62 Efectos de la mutación y cruce sobre el número de generaciones GGAP=0.3

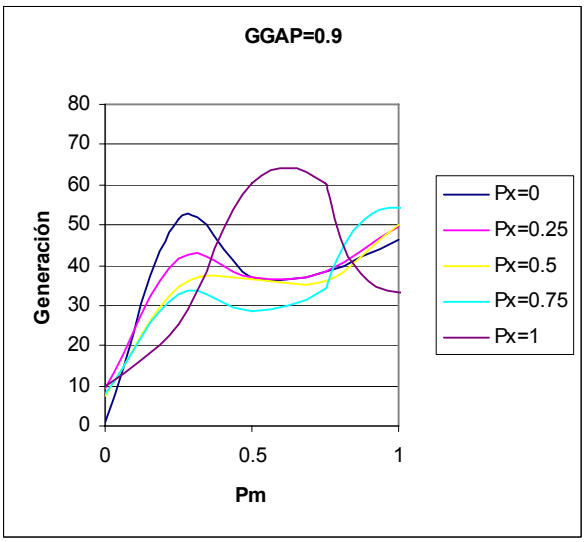


Fig. 63 Efectos de la mutación y cruce sobre el número de generaciones GGAP=0.9

En la Fig. 65 se pone de manifiesto que la dispersión de los valores tiende a disminuir para valores de probabilidad de mutación mayores a 0.5 con una desviación estándar aproximada de 200 (m). Para esos valores de mutación el operador *Px* no parece tener un efecto apreciable sobre la dispersión de los datos.

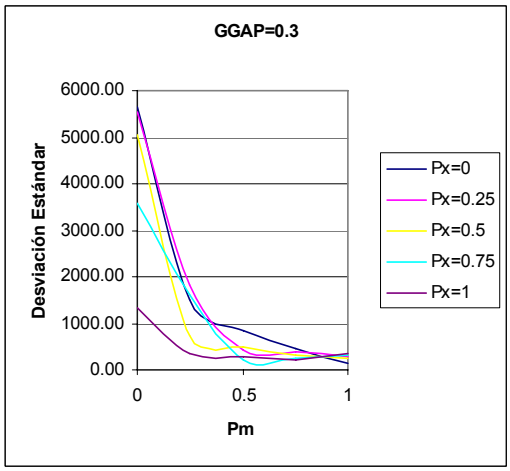


Fig. 64 Efectos de la mutación y cruce sobre la desviación estándar GAP=0.3

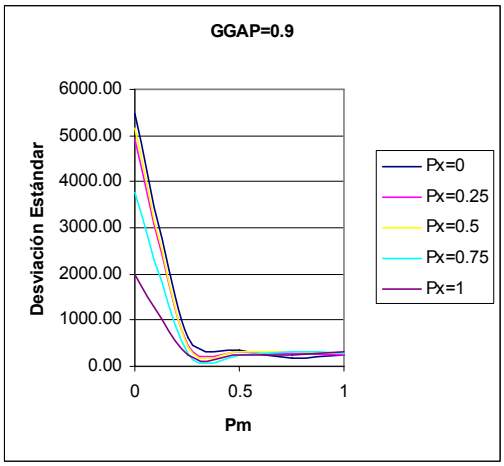


Fig. 65 Efectos de la mutación y cruce sobre la desviación estándar GAP=0.9

- Apoyado en los experimentos descritos, los parámetros elegidos para un recorrido de ocho clientes con tres bloqueos fueron:

- ☑ Probabilidad de mutación: $Pm = 0.5$
- ☑ Probabilidad de cruce: $Px = 0.75$
- ☑ GGAP=0.9.

En la figura 66 se muestra la grafica de la función objetivo contra el número de generaciones. Observe que el algoritmo converge a una buena solución (generación 40) con un valor de Px=0.75 y una probabilidad de mutación igual a 0.5.

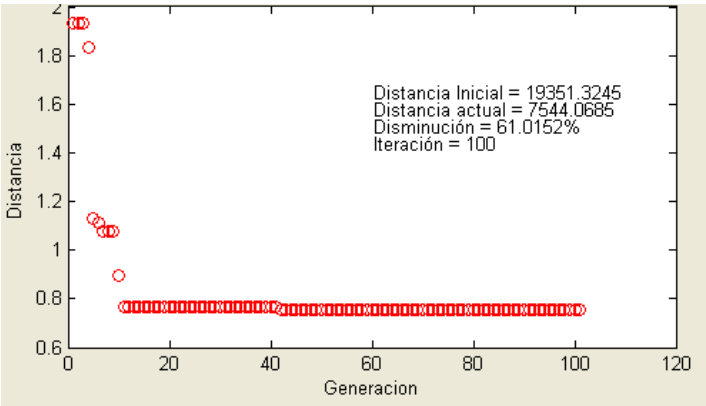


Fig. 66 Gráfica de la función objetivo con Pm=0.5, Px=0.75 y GGAP=0.9.

En la figura 67 se advierte un retardo en la convergencia (generación 60) con un valor bajo de probabilidad de mutación Pm=0.3 y una alta probabilidad de cruce Px=1. Note como la solución encontrada aumentó su valor de distancia (8050.80 m) en comparación con la figura anterior (7544.06 m). Finalmente, el valor óptimo de la función objetivo para este experimento resulto ser en promedio de **μ = 8971.94** [m] con una desviación estándar **σ = 1203.30** [m].

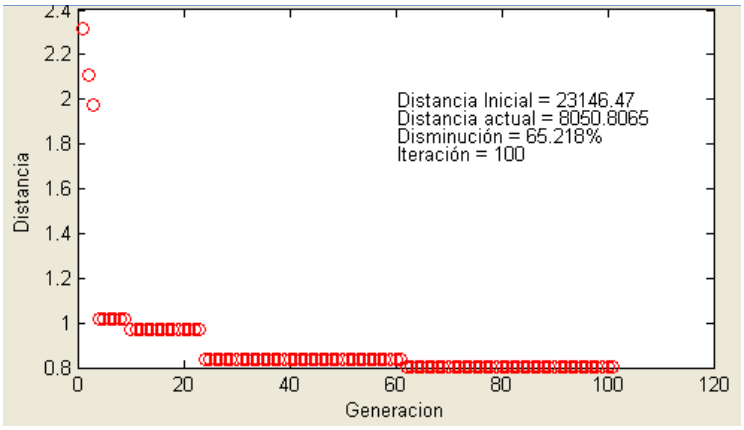


Fig. 67 Gráfica de la función objetivo con Pm=0.3, Px=1 y GGAP=0.9.

El mejor individuo encontrado por el algoritmo resultó ser el representado por el siguiente cromosoma:

A:	1	5	2	6	7	8	4	3	1	1	1	1	1	1	1	7

Recordando la codificación cromosómica, los primeros 8 genes de este individuo representan el orden del recorrido de los clientes y los 8 restantes las rutas alternas.

La **T**abla 9 muestra las rutas disponibles para efectuar el recorrido y el conjunto de calles que pertenecen a cada ruta.

Recorrido	*Ruta disponible*	*Calles de la ruta*
1 – 5	1	18 31 53 79
5 – 2	1	79 86 81 76 64 54 48 49 34 27
2 – 6	1	36 51 52 63 69 85 88 101
6 – 7	1	106 117 116 124
7 – 8	1	135 136 137 138
8 – 4	1	122 105 71
4 – 3	1	14 9
3 – 1	7	9 8 7 6 5

Tabla 14. Orden del recorrido con calles de ruta

En la **F**igura 79 se representa el recorrido óptimo encontrado por el algoritmo para un recorrido de 8 ciudades con 3 bloqueos. Observe que las calles bloqueadas (rectángulos rojos) son evitadas por el algoritmo, en consecuencia el valor de la distancia dinámica es mayor en comparación con el problema libre de bloqueos. Esto se debe a que la penalización de las rutas se incrementa en función del número de las rutas bloqueadas, es decir, un mayor número de bloqueos complica la búsqueda de la ruta óptima si es que existe, por ejemplo, imagine el caso extremo en que todas las calles se encontraran bloqueadas, naturalmente el problema se volvería insoluble y el algoritmo no sería capaz de arrojar una buena solución por muy buena que sea la calibración de sus parámetros.

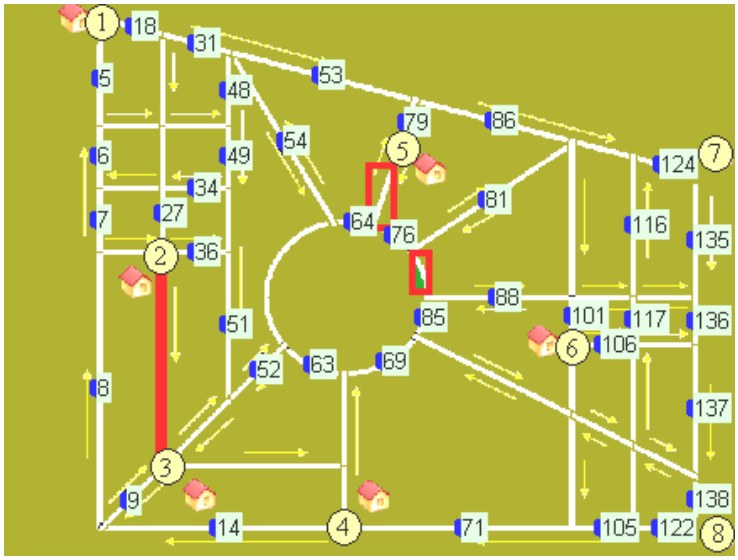


Fig. 68 Secuencia de recorrido óptimo

Recorrido de 4 clientes con bloqueos dinámicos

En este experimento se consideró la velocidad promedio de un cierto flujo vehicular generado aleatoriamente en base al modelo **IDM**. La siguiente figura ilustra la situación, las rutas cuyos vehículos circulan con una *"buena"* velocidad promedio se denotan con un rectángulo verde, y por tanto no penalizan los cromosomas. Por otro lado, un flujo extremadamente lento en cierto instante se consideró como un bloqueo (rectángulo rojo).

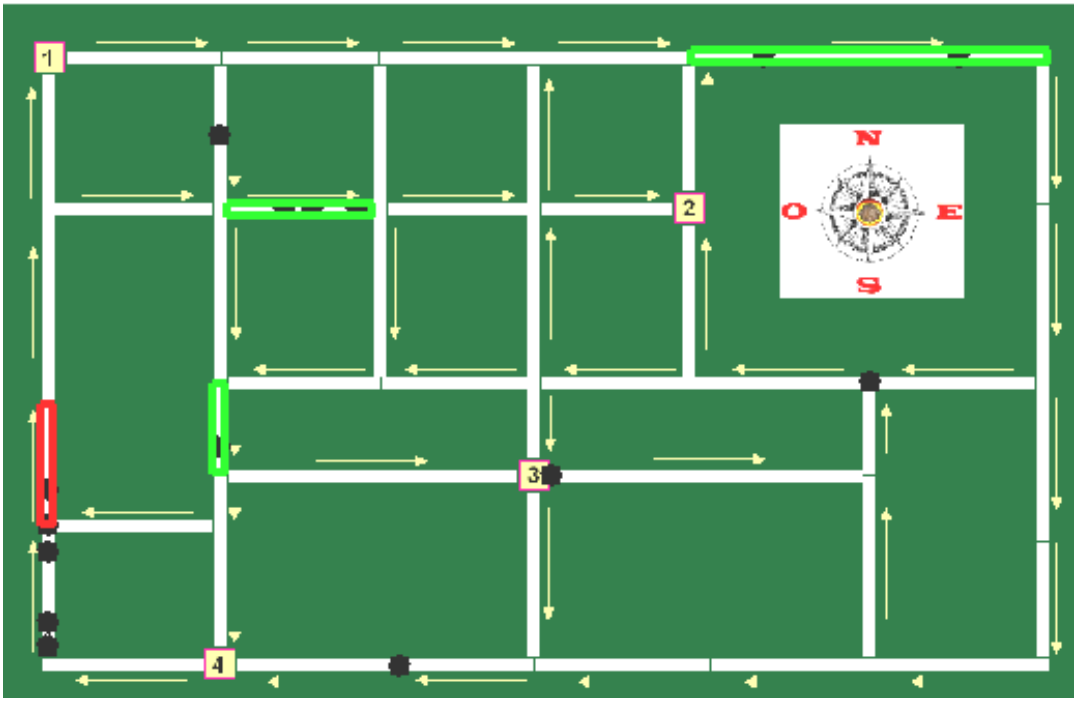


Fig. 69 Mapa de ciudad con bloqueos dinámicos

Dadas las características dinámicas del problema, la ruta óptima será función del tiempo, es decir, las rutas bloqueadas en un instante *t* pueden dejar de estarlo en *t+1*. Por lo tanto la mejor solución dependerá de las circunstancias cambiantes del entorno. Por tal motivo, la calibración de los parámetros resulta compleja y laboriosa. No obstante, los resultados de la experimentación demuestran que el algoritmo entrega buenas soluciones empleando calibraciones similares a las utilizadas en los casos estáticos. Por lo que la probabilidad de mutación se fijó en 0.5 y el cruce PMX en 0.75 con un GGAP=0.9.

La Figura 70 muestra los valores de la función objetivo vs el número de generaciones, la gráfica fue obtenida en una simulación con un tráfico de 200 vehículos. Como se puede advertir, el valor óptimo depende de las características dinámicas de los bloqueos que se presentan en el flujo vehicular. Por ejemplo, en las primeras generaciones el flujo presentó una buena velocidad promedio (ruta libre), sin embargo, aproximadamente en la generación 20 la velocidad de algunos vehículos disminuyó lo suficiente como para ser considerados como bloqueos. En consecuencia, se penalizó la función objetivo y se incrementó la *distancia dinámica* de la ruta encontrada hasta el momento. Finalmente, para la generación 80 el algoritmo encuentra una buena solución considerando dichas situaciones imprevistas de bloqueo.

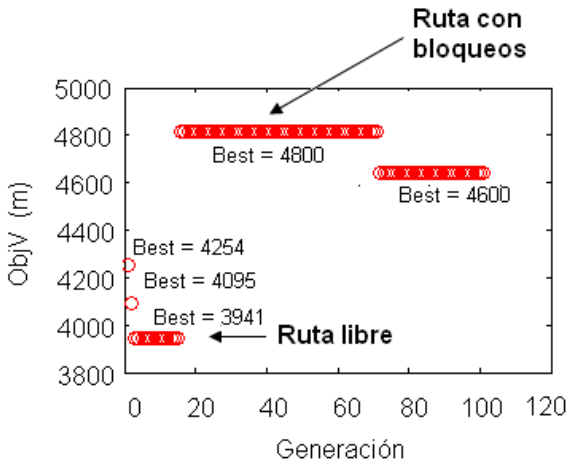


Fig. 70 Gráfica de la Función Objetivo vs. Número de generación

Este experimento permitió observar la capacidad de respuesta del algoritmo, que gracias a su rapidez de convergencia hacia soluciones óptimas demostró ser aplicable a situaciones donde se requiere conocer ambientes en tiempo real.

Resultados de la experimentación

Se observó que el efecto del operador de cruce **PMX** en la búsqueda combinatoria de la ruta óptima, fue inferior al efecto que tuvo el operador de mutación. La determinación del valor óptimo de la probabilidad de mutación resultó ser mucho más crucial que el relativo a la probabilidad de cruce.

Se obtuvieron resultados muy satisfactorios utilizando una evolución primitiva, es decir, considerando únicamente el proceso de selección y el operador de mutación.

Se apreció que la complejidad del problema se agranda conforme aumenta el número de ciudades (*NVAR*), afectando la rapidez de convergencia del algoritmo.

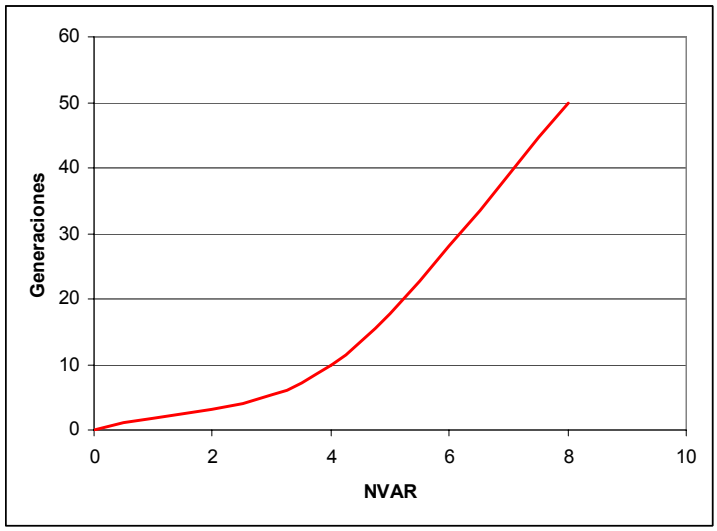


Fig.71 Número de generaciones vs. Número de ciudades

En resumen, algunas observaciones obtenidas fueron:

- El uso de porcentajes bajos de mutación (menores a 0.2) no mejoró el desempeño del algoritmo.
- El uso de porcentajes de GGAP menores a 0.8 no mejoró el desempeño del algoritmo.
- La mutación tuvo mayor importancia que el operador de cruce para encontrar la solución óptima, sin embargo el cruce en combinación con la mutación aumentaron la rapidez de convergencia

CONCLUSIONES

Considerando que el objetivo principal del presente trabajo, fue diseñar un algoritmo híbrido para encontrar la ruta óptima en un flujo vehicular simulado en el cual se añaden situaciones imprevistas de bloqueo, se comprobó experimentalmente que el tratamiento propuesto es viable para resolver dicho problema. Cabe destacar que el problema implica una mayor complejidad combinatoria que otros ampliamente estudiados (por ejemplo, el *Problema del Agente Viajero*). En efecto, la obtención de una solución supone no sólo encontrar un recorrido factible de los destinos, sino también la selección dinámica de la ruta óptima dentro de un amplio rango de posibles rutas.

Abordar este asunto con métodos evolutivos, se hace plausible y atractivo debido a que en la actualidad se dispone de computadoras más eficientes. Idea que no era fácil de realizarse en el pasado, es decir, estas técnicas se hubiesen desechado por incosteables ya que requieren de muchos recursos computacionales.

En el diseño del algoritmo, se propuso una codificación cromosómica entera, lo que permitió representar de forma más natural la solución del problema ya que no se necesitó decodificar cada parámetro para evaluar la función objetivo. Sin embargo, este tipo de codificación requirió definir operadores genéticos específicos, es el caso del cruce PMX.

La cuestión de los parámetros del algoritmo (probabilidad de cruce y mutación, GGAP, etc.) fue una parte medular del diseño. Al igual que la mayoría de los trabajos sobre algoritmos genéticos, se basó en la calibración empírica hasta obtener un buen funcionamiento del algoritmo. Se adolece de una teoría más robusta que brinde una fundamentación más sólida sobre el funcionamiento de este tipo de algoritmos, esto permitiría en particular, poder predecir su rendimiento.

Para analizar el comportamiento del algoritmo se emplearon herramientas de la estadística descriptiva, debido a que no existen resultados obtenidos por otros algoritmos para utilizar técnicas comparativas. El algoritmo en la mayoría de los casos, encuentra soluciones aproximadas que no necesariamente son las óptimas, sin embargo, los resultados de la experimentación en el presente estudio, demuestran que dichas soluciones son bastante aceptables al problema con baja dispersión y rápida convergencia.

De acuerdo a la discusión expuesta en este trabajo, los algoritmos, especialmente los genéticos, no son una panacea para resolver cualquier tipo de problemas; lo que si existe es una gran cantidad de algoritmos que son susceptibles de adaptarse a las características de la situación que se trata de resolver.

Para un futuro estudio se recomienda incorporar un localizador de posicionamiento global (**GPS**) para detectar bloqueos en tiempo real. Así mismo, el algoritmo podría ser complementado con mapas satelitales con contenido dinámico, por ejemplo, congestión vehicular, condiciones meteorológicas, eventos impredecibles locales, etc.

Es importante señalar que es conveniente llevar a cabo más pruebas para refinar los parámetros del algoritmo desarrollado y usar máquinas más potentes para aprovechar su capacidad computacional, y de esta manera mejorar los resultados mostrados en este trabajo.

Finalmente, quedan abiertas futuras líneas de investigación, es el caso de utilizar técnicas alternativas dentro de la *Inteligencia Artificial*, como redes neuronales o lógica difusa, para comparar los criterios y la calidad en el resultado de la optimización de rutas de vehículos presentada.

REFERENCIAS

[1] Chatterjee, Sangit and Carrera, Cecilia and Lynch, Lucy A. (1996). ***Genetic algorithms and traveling salesman problems***. European Journal of Operational Research. V. 93. p.p. 490-510.

[2] Ahmed, E. and Elettreby, M.F. (2006). ***On combinatorial optimization motivated by biology***. Applied Mathematics and Computation. V. 172. p.p. 40-48.

[3] Baker, Barrie M. and Ayechew, M. A. (2003). ***A genetic algorithm for the vehicle routing problem***. Computers & Operations Research. V. 30. p.p. 787-800.

[4] Cai, Z. and Peng, J. and Gao, W. and Wei, W. and Kang, L. (2004). ***A novel Evolutionary Algorithm and its application for the traveling salesman problem***. WCICA 2004 – Fifth World Congress on Intelligent Control and Automation, Conference Proceedings. V. 3. p.p. 2253-2257.

[5] W. Banzhaf (1990). ***The molecular traveling salesman***, Biological Cybernetics, p.p. 64, 7-14.

[6] C. Darwin (1859). ***On the Origin of Species by Means of Natural Selection***, Murray, London.p.p 76-89.

[7] L. Davis (ed.) (1991). ***Handbook of Genetic Algorithms***, Van Nostrand Reinhold, New York.p.p 123-187.

[8] J. E. Baker, "***Reducing bias and inefficiency in the selection algorithm",*** *Proc ICGA 2*, pp. 14-21, Lawrence Erlbaum Associates, Publishers, 1987.

[9] L. Davis (1985). ***Applying adaptive algorithms to epistatic domains***, Proceedings of the International Joint Conference on Arti_cial Intelligence, 162-164.

[10] J. David Schaffer, Rich Caruana, Larry J. Eshelman, Rajarshi Das (1989) ***A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization***. ICGA p.p. 51-60

[11] D. B. Fogel (1993). ***Applying evolutionary programming to selected traveling salesman problems,*** Cybernetics and Systems,p.p. 24-36.

[12] D. E. Goldberg, Jr. R. Lingle (1985). ***Alleles, loci and the traveling salesman problem***, en Proceedings of the First International Conference on Genetic Algorithms and Their Applications, p.p.154-159.

[13] Holland, J.H., ***Adaptation in Natural and Artificial Systems,*** Ann Arbor, MA: University of Michigan (1975).

[14] Michalewics,Z, *Genetic* ***Algorithm ÷ Data Structure = Evolution*** *Programs,* Springer- Verlag, second edition (1994).

[15] Chartrand, G. and L. Lesniak, ***Graphs and Digraphs,*** Wadsworth & Brooks, Monterey,1979.

[16] Laporte, G., **The traveling salesman problem: an overview of exact and approximate algorithms**, *European Journal of Operational Research* 59, 231-247, 1992.

[17] Coello, Carlos. (2000). ***An updated survey of GA-based multiobjective optimization techniques.*** *ACM Computing Surveys*, vol.32, no.2, p.p.109-143.
[18] Forrest, Stephanie. (1993) ***Genetic algorithms: principles of natural selection applied to computation.*** *Science*, vol.261, p.p. 872-878.
[19] Martin Treiber, (2007) ***Congested Traffic States in Empirical Observations and Microscopic Simulations.*** Institute of Theoretical Physics, University of Stuttgart, Pfaffenwaldring 57, D-70550 Stuttgart, Germany
[20] Clander R., Herman E. Y Montroll E. (1998) ***Traffic Dynamics: Studies in car Following.*** Operational Research. S.l. no.2; p.p 41-43.
[21] Chipperfield, A.J. and Fleming, P.J., 1995. ***The MATLAB Genetic Algorithm Toolbox*****,** IEE Colloqium on Applied Control Techniques Using MATLAB, Digest No. 1995/014.
[22] Alba, E., & Dorronsoro, B. (2004). ***Solving the vehicle routing problem by using cellular genetic algorithms*****.** In Lecture notes in computer science (Vol. 3004, pp. 11–20).
[23] Osman, I. H., & Kelly, J. P. (1996**).** ***Meta-heuristics: An overview.*** In I.
[24] H. Osman & J. P. Kelly (Eds.), ***Meta-heuristics: Theory and applications*** (pp.1–21). Boston: Kluwer Academic Publishers.
[25] D.H. Wolpert**,** W.G. Macready, ***No free lunch theorems for search***, Tech**.** Rep**.** SFI**-**TR**-**95-02-010**,** Santa Fe**,** NM**,** 1995.

ANEXOS

A.1 Operador de cruce PMX implementado en MATLAB

```
%Operador de cruce PMX (Partially Matched Crossover)

function NewChrom = PMX(Chrom,Px)

% Identificar el tamaño de la población Nind y la longitud de los
% individuos Lind
[Nind,Lind] = size(Chrom);
hijos=zeros(Nind,Lind); % Descendencia
regionP1=zeros(1,Lind); % region de cruce padre1
regionP2=zeros(1,Lind); % region de cruce padre2
% Puntos de cruce aleatorios
aleatorio=randperm(Lind);
p1=aleatorio(1);
p2=aleatorio(2);
hijos=Chrom;

if rand(1)<=Px, % si es menor que la probabilidad de cruce ejecutar PMX
while p1>p2
aleatorio=randperm(Lind);
p1=aleatorio(1);
p2=aleatorio(2);
end

for i=1:2:Nind-1
   % Copiar información directa a los hijos
   for w=1:Lind
   hijos(i,w)=Chrom(i,w);
   hijos(i+1,w)=Chrom(i+1,w);
   end
   % definir región de cruce PMX para el padre 1
   for w=p1:p2
   regionP1(w)=Chrom(i,w);
   end
    % definir región de cruce PMX para el padre 2
   for w=p1:p2
   regionP2(w)=Chrom(i+1,w);
   end
   % copiar las regiones de cruce en los hijos
   for w=p1:p2
   hijos(i,w)=regionP2(w);
   hijos(i+1,w)=regionP1(w);
```

```
 end
 % Identificar alelos repetidos en el hijo 1
for q=p1:p2,
     for w=1:p1 %Alelo repetido en el hijo 1?
       if regionP2(q)==hijos(i,w),
         % cambiar por alelo padre 2
          for r=1:Lind
            esta=0;
            for u=1:Lind
               % Está el alelo en el cromosoma?
              if Chrom(i+1,r)==hijos(i,u),esta=1;end
            end
            if esta==0, hijos(i,w)=Chrom(i+1,r);end
          end
       end
    end
end
for q=p1:p2,
     for w=p2:Lind %Alelo repetido en el hijo 1?
       if regionP2(q)==hijos(i,w),
         % cambiar por alelo padre 2
          for r=1:Lind
            esta=0;
            for u=1:Lind
               % Está el alelo en el cromosoma?
              if Chrom(i+1,r)==hijos(i,u),esta=1;end
            end
            if esta==0, hijos(i,w)=Chrom(i+1,r);end
          end
       end
    end
 end
% Identificar alelos repetidos en el hijo 2
for q=p1:p2,
    for w=1:p1
          %Alelo repetido en el hijo 2?
       if regionP1(q)==hijos(i+1,w),
          % cambiar por alelo padre 1
          for r=1:Lind
            esta=0;
            for u=1:Lind
              if Chrom(i,r)==hijos(i+1,u),esta=1;end
            end
            if esta==0, hijos(i+1,w)=Chrom(i,r);end
```

```
            end
          end
        end
    end

    for q=p1:p2,
        for w=p2:Lind
              %Alelo repetido en el hijo 2?
          if regionP1(q)==hijos(i+1,w),
              % cambiar por alelo padre 1
            for r=1:Lind
              esta=0;
              for u=1:Lind
                if Chrom(i,r)==hijos(i+1,u),esta=1;end
              end
              if esta==0, hijos(i+1,w)=Chrom(i,r);end
            end
          end
        end
    end

end
   % si la población es impar copia directamente el cromosoma sin aparear
   if  rem(Nind,2)~=0,
      hijos(Nind,:)=Chrom(Nind,:);
   end
end

NewChrom=hijos; % Actualiza los cromosomas
end % final de función
NIND =50;          % Numero de individuos por generacion
```

A.2 Problema del agente viajero implementado en MATLAB

```
MAXGEN = 200;       % Numero maximo de generaciones
GGAP = 0.9;         % GAP generacional
NVAR = 16;          % Numero de ciudades

% Poblacion inicial


Chrom=zeros(NIND,NVAR);

 for i=1:NIND,
  buffer=randperm(NVAR);
       for j=1:NVAR,
           Chrom(i,j)=buffer(j);
       end
 end

% Matriz de distancias
  load D

% Reiniciar contadores
  Best = NaN*ones(MAXGEN,1);   % El mejor individuo de la poblacion actual
  gen = 0;                     % contador generacional

% Evaluar la poblacion inicial
  ObjV = objviajero(Chrom,D);

% Graficar convergencia
 Best(gen+1) = min(ObjV);
     plot(Best,'ro'); xlabel('Generación'); ylabel('Distancia');
     text(0.5,0.90,['Distancia actual = ', num2str(Best(gen+1))],'Units','normalized');
     text(0.5,0.95,['Distancia Inicial = ', num2str(Best(1))],'Units','normalized');
     dism=100-(Best(gen+1)/Best(1))*100;
     text(0.5,0.85,['Disminución = ', num2str(dism),'%'],'Units','normalized');
     text(0.5,0.80,['Iteración = ', num2str(gen)],'Units','normalized');

  drawnow;

  % Loop generacional
  while gen < MAXGEN,
```

```
% asigna valor de fitness a toda la poblacion
   FitnV = ranking(ObjV);

% Selecciona individuos
   SelCh = select('sus', Chrom, FitnV, GGAP);


% Xover
SelCh=PMX(SelCh,0.6);

% Mutacion

SelCh = mutacion(SelCh,0.8);



 % Evaluar la descendencia, llama la funcion objetivo
   ObjVSel = objviajero(SelCh,D);

 % Reinserta la descendencia en la poblacion actual
   [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);

 % Incrementa el contador generacional
   gen = gen+1;

 % Actualiza el grafico y muestra el mejor individuo
   Best(gen+1) = min(ObjV);
   plot(Best,'ro'); xlabel('Generación'); ylabel('Distancia');
   text(0.5,0.90,['Distancia actual = ', num2str(Best(gen+1))],'Units','normalized');
   text(0.5,0.95,['Distancia Inicial = ', num2str(Best(1))],'Units','normalized');
   dism=100-(Best(gen+1)/Best(1))*100;
   text(0.5,0.85,['Disminución = ', num2str(dism),'%'],'Units','normalized');
   text(0.5,0.80,['Iteración = ', num2str(gen)],'Units','normalized');

   drawnow;
 % guardar mejor recorrido

 [ordenados, indx] = sort(ObjV,1);

  recorrido=Chrom(indx(1),:)

end
```

A.3 Optimización de rutas dinámico con bloqueos implementado en MATLAB

```
function pushbutton1_Callback(hObject, eventdata, handles)

% cargar imagen mapa
axes(handles.axes2);
axis off
img=imread('mapa.bmp');
im_g=rgb2gray(img);
umb=graythresh(im_g);
bw=im2bw(im_g,umb);
imshow(img);
[L Ne]=bwlabel(bw);
propied=regionprops(L,'all');
hold on;

GGAP = str2double(get(handles.GGAP, 'String'));   % GAP generacional
MAXGEN=str2double(get(handles.MAXGEN, 'String')); % Numero maximo de
generaciones
NIND=str2double(get(handles.NIND, 'String'));     % Numero de individuos
Pm=str2double(get(handles.Pm, 'String'));         % Probabilidad de mutacion
Px=str2double(get(handles.Px, 'String'));         % Probabilidad de cruce PMX
pb=str2double(get(handles.Pb, 'String'));         % probabilidad de bloqueo
NVAR=str2double(get(handles.NVAR, 'String'));     % Numero de Ciudades


load rutas;

bandera=0;
apagar=0;
ciclos=0;


if rand(1)<=pb,

axes(handles.axes2);
rectangle('Position', propied(83).BoundingBox,'EdgeColor','r','LineWidth',3);
drawnow;
pause(0.5)
rectangle('Position', propied(28).BoundingBox,'EdgeColor','r','LineWidth',3);
drawnow;
pause(0.5)
rectangle('Position', propied(74).BoundingBox,'EdgeColor','r','LineWidth',3);
```

```
drawnow;

end



% Poblacion inicial aleatoria

Chrom=zeros(NIND,NVAR*2);

ch=zeros(NIND,NVAR);

 for i=1:NIND,
   buffer=randperm(NVAR);
        for j=1:NVAR,
            Chrom(i,j)=buffer(j);
        end
   buffer=randperm(NVAR);
        for j=NVAR+1:NVAR*2,
            Chrom(i,j)=buffer(j-NVAR);
        end
 end

 Chrom

% Reiniciar contadores
   Best = NaN*ones(MAXGEN,1);   % El mejor individuo de la poblacion actual
   gen = 0;         % contador generacional


% Reparar  cromosomas
   Chrom=reparador(Chrom,rutas);


% cargar rutas disponibles
for i=1:NIND,
    for j=NVAR+1:NVAR*2,
        ch(i,j-NVAR)=Chrom(i,j);
    end
end


% Evaluar la poblacion inicial
   ObjV = objviajero(Chrom,rutas, propied,bandera);
```

```
% Detectar Bloqueo

 if rand(1)<=pb&&apagar==0,
 bandera=1 ;
 axes(handles.axes2);
 Chrom=bloqueo(Chrom,propied,NVAR);
 apagar=1;

 end


% Graficar convergencia
 Best(gen+1) = min(ObjV);
 axes(handles.axes1);
 plot(Best,'ro'); xlabel('Generacion'); ylabel('Distancia ');
 text(0.5,0.70,['Distancia actual = ', num2str(Best(gen+1))],'Units','normalized');
 text(0.5,0.75,['Distancia Inicial = ', num2str(Best(1))],'Units','normalized');
    dism=100-(Best(gen+1)/Best(1))*100;
 text(0.5,0.65,['Disminución = ', num2str(dism),'%'],'Units','normalized');
 text(0.5,0.60,['Iteración = ', num2str(gen)],'Units','normalized');
 drawnow;


 % Loop generacional
while gen < MAXGEN,


 % asigna valor de fitness a toda la poblacion
    FitnV = ranking(ObjV);


 % Selecciona individuos
    SelCh = select('rws', Chrom, FitnV, GGAP);


 % cruce PMX de orden de recorridos
 for tam1=1:size(SelCh,1)
  for tam2=1:size(SelCh,2)/2
   pmx(tam1,tam2)=SelCh(tam1,tam2);
  end
 end

 pmx=PMX(pmx,Px);
```

```
    % insertar descendencia en la poblacion actual
    for tam1=1:size(SelCh,1)
      for tam2=1:size(SelCh,2)/2
       SelCh(tam1,tam2)=pmx(tam1,tam2);
      end
    end
    % Mutacion
     SelCh = mutacion(SelCh,Pm);

     % Reparar  cromosomas
     SelCh=reparador(SelCh,rutas);


     save chrom

% cargar rutas disponibles
for i=1:size(SelCh,1),

    for j=NVAR+1:NVAR*2,
        ch(i,j-NVAR)=SelCh(i,j);
    end
end



    % Evaluar la descendencia, llama la funcion objetivo
     ObjVSel = objviajero(SelCh,rutas,propied,bandera);

    % Reinserta la descendencia en la poblacion actual
      [Chrom ObjV]=reins(Chrom,SelCh,1,1,ObjV,ObjVSel);

    % Incrementa el contador generacional
      gen = gen+1;

    % Actualiza el grafico y muestra el mejor individuo
      Best(gen+1) = min(ObjV);
   axes(handles.axes1);
   plot(Best,'ro'); xlabel('Generacion'); ylabel('Distancia ');
   text(0.5,0.70,['Distancia actual = ', num2str(Best(gen+1))],'Units','normalized');
   text(0.5,0.75,['Distancia Inicial = ', num2str(Best(1))],'Units','normalized');
      dism=100-(Best(gen+1)/Best(1))*100;
   text(0.5,0.65,['Disminución = ', num2str(dism),'%'],'Units','normalized');
   text(0.5,0.60,['Iteración = ', num2str(gen)],'Units','normalized');
```

```
    drawnow;

     % guardar mejor recorrido y ruta
     [ordenados, indx] = sort(ObjV,1);

     sec=Chrom(indx(1),:)

        for j=NVAR+1:NVAR*2,
         sch(j-NVAR)=Chrom(indx(1),j);
        end

end
```

A.4 Codificación de mapa de ciudad (arreglo tipo célula)

```
rutas(1,1)={[]};
rutas(1,2)={[18 25 26 27 0 0 0 0 ;...
        18 31 48 49 34 27 0 0;...
        18 25 33 49 34 27 0 0]};
rutas(1,3)={[18 31 54 60 52 35 0 0 0 0 0;18 31 48 49 50 51 35 0 0 0          0;...
        18 25 33 49 50 51 35 0 0 0 0;...
        18 31 53 86 81 76 64 60 52 35 0;...
        18 31 54 64 76 83 85 69 63 52 35]};
rutas(1,4)={[18 31 54 64 76 83 85 82 103 71 0 0 0 0 0;...
        18 31 53 86 81 83 85 82 103 71 0 0 0 0 0;...
        18 31 53 86 100 88 85 82 103 71 0 0 0 0 0;...
        18 31 53 86 100 107 123 136 137 121 104 103 71 0 0;...
        18 31 54 64 76 83 88 107 123 136 137 121 104 103 71]};
rutas(1,5)={[18 31 53 79 0 0 0 0 0 0 0 0 0 0 0;...
        18 31 54 64 74 0 0 0 0 0 0 0 0 0 0;...
        18 31 53 86 81 76 74 0 0 0 0 0 0 0 0;...
        18 31 53 86 100 88 83 76 74 0 0 0 0 0 0;...
        18 25 33 49 50 51 52 60 64 74 0 0 0 0 0]};
rutas(1,6)={[18 31 53 86 100 101 0 0 0 0 0 0 0 0 0;...
        18 31 54 64 76 83 88 101 0 0 0 0 0 0 0;...
        18 31 54 64 76 81 100 101 0 0 0 0 0 0 0;...
        18 31 48 49 50 51 52 63 69 85 88 101 0 0 0;...
        18 25 33 49 50 51 52 63 69 85 88 101 0 0 0]};
rutas(1,7)={[18 31 53 86 108 124 0 0 0 0 0 0 0 0 0;...
        18 31 54 64 76 81 108 124 0 0 0 0 0 0 0;...
        18 31 53 86 100 107 116 124 0 0 0 0 0 0 0;...
        18 31 54 64 76 83 88 107 116 124 0 0 0 0 0;...
        18 31 48 49 50 51 52 60 64 76 81 108 124 0 0 ]};
rutas(1,8)={[18 31 53 86 100 107 123 136 137 138 0 0 0 0 0;...
        18 31 54 64 76 83 85 82 104 121 138 0 0 0 0;...
        18 31 53 86 81 83 85 82 104 121 138 0 0 0 0;...
        18 31 54 64 76 81 100 107 123 136 137 138 0 0 0;...
        18 31 54 64 76 83 88 107 123 136 137 138 0 0 0]};
rutas(2,1)={[36 51 52 60 54 48 49 34 12 6 5 0 0 0 0 0;...
        36 51 52 63 69 85 83 76 64 54 48 49 34 12 6 5]};
rutas(2,2)={[]};
rutas(2,3)={[28 0 0 ; 36 51 35 ]};
rutas(2,4)={[36 51 52 63 69 82 103 71 0 0 0;...
        36 51 52 60 64 76 83 85 82 103 71]};
rutas(2,5)={[36 51 52 60 64 74 0 0 0 0 0 0 0 0 0;...
```

```
        36 51 52 63 69 85 83 76 74 0 0 0 0 0;...
        36 51 52 63 69 82 104 118 117 107 88 83 76 74 ]};
rutas(2,6)={[36 51 52 63 69 85 88 101 0 0 0;...
        36 51 52 60 64 76 83 88 101 0 0;...
        36 51 52 60 54 53 86 100 101 0 0;...
        36 51 52 60 64 76 81 100 101 0 0;...
        36 51 52 63 69 82 104 118 117 107 101]};
rutas(2,7)={[36 51 52 60 64 76 81 108 124 0 0 0;...
        36 51 52 63 69 82 104 118 117 116 124 0; ...
        36 51 52 63 69 85 88 107 116 124 0 0;...
        36 51 52 60 54 53 86 108 124 0 0 0;...
        36 51 52 63 69 85 83 81 100 107 116 124]};
rutas(2,8)={[36 51 52 63 69 82 104 121 138 0 0 0 0;...
        36 51 52 63 69 85 88 107 123 136 137 138 0;...
        36 51 52 60 64 76 83 88 107 123 136 137 138;...
        36 51 52 60 64 76 81 100 107 123 136 137 138]};
rutas(3,1)={[9 8 7 6 5 0 0 0 0 0 0 0 0 0;...
        35 52 60 54 48 49 34 12 6 5 0 0 0 0 0;...
        38 67 63 60 54 48 49 34 12 6 5 0 0 0 0;...
        38 67 69 85 83 76 64 54 48 49 34 12 6 5 0;...
        35 52 63 69 85 83 76 64 54 48 49 34 12 6 5]};
rutas(3,2)={[9 8 13 0 0 0 0 0 0 0 0 0 0;...
        35 52 60 64 54 49 34 27 0 0 0 0 0;...
        38 67 63 60 54 48 49 34 27 0 0 0 0;...
        38 67 69 85 83 76 64 54 48 49 34 27 0;...
        35 52 63 69 85 83 76 64 54 48 49 34 27]};
rutas(3,3)={[]};
rutas(3,4)={[35 52 63 69 82 103 71 0 0 0 0 0 0 0 0;...
        38 67 69 82 103 71 0 0 0 0 0 0 0 0 0;...
        35 52 60 64 76 83 85 82 103 71 0 0 0 0 0;...
        35 52 60 64 76 83 88 107 123 136 137 121 104 103 71]};
rutas(3,5)={[35 52 60 64 74 0 0 0 0 0 0 0 0 0;...
        38 67 63 60 64 74 0 0 0 0 0 0 0 0;...
        38 67 69 85 83 76 74 0 0 0 0 0 0 0;...
        35 52 63 69 85 83 76 74 0 0 0 0 0;...
        9 8 7 6 11 33 49 50 51 52 60 64 74]};
rutas(3,6)={[38 67 69 85 88 101 0 0 0;...
        35 52 63 69 85 88 101 0 0;...
        35 52 60 64 76 83 88 101 0;...
        38 67 63 60 64 76 83 88 101]};
rutas(3,7)={[35 52 60 64 76 81 108 124 0;...
        35 52 63 69 85 83 81 108 124;...
        38 67 69 85 83 81 108 124 0;...
        38 67 69 85 88 107 116 124 0;...
```

```
        38 67 69 82 104 118 117 116 124]};
rutas(3,8)={[35 52 63 69 85 82 104 121 138 0 0;...
        38 67 69 85 82 104 121 138 0 0 0;...
        35 52 63 69 85 88 107 123 136 137 138;...
        38 67 69 85 88 107 123 136 137 138 0]};
rutas(4,1)={[14 8 7 6 5 0 0 0 0 0 0 0 0 0;...
        68 67 69 85 83 76 64 54 48 49 34 12 6 5;...
        68 67 63 60 54 48 49 34 12 6 5 0 0 0 ]};
rutas(4,2)={[68 67 63 60 54 48 49 34 27 0 0 0 0;...
        68 67 69 85 83 76 54 48 49 34 27 0 0]};
rutas(4,3)={[14 9 0 0 0 0 0 0 0 0 0 0 0;...
        68 67 63 52 35 0 0 0 0 0 0 0 0;...
        68 67 69 85 83 76 64 60 52 35 0 0 0;...
        68 67 69 85 83 76 64 54 48 49 50 51 35]};
rutas(4,4)={[]};
rutas(4,5)={[68 67 69 85 83 76 74 0 0 0 0 0 0 0;...
        68 67 63 60 64 74 0 0 0 0 0 0 0 0;...
        14 9 8 7 6 11 33 49 50 51 52 60 64 74]};
rutas(4,6)={[68 67 69 85 88 101 0 0 0;...
        68 67 69 82 104 118 117 107 101;...
        68 67 63 60 64 76 81 100 101]};
rutas(4,7)={[68 67 69 85 83 81 108 124 0 0 0 0 0 0 0 0 0;...
        68 67 69 85 88 107 116 124 0 0 0 0 0 0 0 0 0;...
        68 67 69 82 104 118 117 116 124 0 0 0 0 0 0 0 0;...
        68 67 63 60 64 76 81 108 124 0 0 0 0 0 0 0 0;...
        14 8 7 6 11 33 49 50 51 52 63 69 85 83 81 108 124;...
        14 8 7 6 11 33 49 50 51 52 60 64 76 81 108 124 0]};
rutas(4,8)={[68 67 69 82 104 121 138 0 0 0 0 0 0 0 0 0;...
        68 67 69 85 88 107 123 136 137 138 0 0 0 0 0;...
        68 67 63 60 64 76 83 88 107 123 136 137 138 0 0 0;...
        68 67 63 60 64 76 83 85 82 104 121 138 0 0 0 0;...
        14 8 7 6 11 33 49 50 51 52 63 69 82 104 121 138]};
rutas(5,1)={[74 64 54 48 49 34 12 6 5 0 0 0 0 0 0;...
        74 76 83 85 69 63 60 54 48 49 34 12 6 5 0 0;...
        74 76 81 100 88 85 69 63 60 54 48 49 34 12 6 5;...
        79 86 81 76 64 54 48 49 34 12 6 5 0 0 0 0;...
        79 86 81 83 85 69 63 60 54 48 49 34 12 6 5 0]};
rutas(5,2)={[79 86 81 76 64 54 48 49 34 27 0 0 0 0 0 0;...
        79 86 81 83 85 69 63 60 54 48 49 34 27 0 0 0 ]};
rutas(5,3)={[74 64 60 52 35 0 0 0 0 0;...
        74 76 83 85 69 63 52 35 0 0;...
        74 64 54 48 49 50 51 35 0 0;...
        79 86 81 83 85 69 63 52 35 0;...
        79 86 81 76 64 60 52 35 0 0;...
```

```
        79 86 100 88 85 69 63 52 35 0;...
        79 86 100 88 83 76 64 60 52 35]};
rutas(5,4)={[74 76 83 85 82 103 71 0 0 0 0 0;...
        74 76 81 100 107 123 136 137 121 104 103 71;...
        79 86 100 107 123 136 137 121 104 103 71 0;...
        79 86 81 83 85 82 103 71 0 0 0 0]};
rutas(5,5)={[]};
rutas(5,6)={[74 76 83 88 101 0 0 0 0 0 0;...
        74 76 83 85 82 104 118 117 107 101 0;...
        74 64 60 63 69 85 88 101 0 0 0 ;...
        74 64 60 63 69 82 104 118 117 107 101;...
        79 86 100 101 0 0 0 0 0 0 0;...
        79 86 81 83 88 101 0 0 0 0 0]};
rutas(5,7)={[74 76 81 108 124 0 0 0 0 0 0 0;...
        74 64 54 53 86 108 124 0 0 0 0 0;...
        74 76 83 88 107 116 124 0 0 0 0 0;...
        74 64 60 63 69 85 83 81 108 124 0 0;...
        74 76 83 85 69 63 60 54 53 86 108 124;...
        74 64 60 63 69 85 88 107 116 124 0 0;...
        79 86 108 124 0 0 0 0 0 0 0 0;...
        79 86 81 83 88 107 116 124 0 0 0 0]};
rutas(5,8)={[74 76 83 85 82 104 121 138 0 0 0 0 0 0 0 0;...
        74 76 83 88 107 123 136 137 138 0 0 0 0 0 0 0;...
        74 64 54 53 86 100 107 123 136 137 138 0 0 0 0 0;...
        74 64 60 63 69 82 104 121 138 0 0 0 0 0 0 0;...
        74 64 60 63 69 85 88 107 123 136 137 138 0 0 0 0;...
        74 76 83 85 69 63 60 54 53 86 100 107 123 136 137 138;...
        79 86 100 107 123 136 137 138 0 0 0 0 0 0 0 0]};
rutas(6,1)={[102 82 85 83 76 64 54 48 49 34 12 6 5 0 0 0 0;...
        102 82 69 63 60 54 48 49 34 12 6 5 0 0 0 0 0;...
        106 120 137 121 104 82 85 83 76 64 54 48 49 34 12 6 5;...
        106 120 137 121 104 82 69 63 60 54 48 49 34 12 6 5 0;...
        106 117 107 88 83 76 64 54 48 49 34 12 6 5 0 0 0 ;...
        106 117 107 88 85 69 63 60 54 48 49 34 12 6 5 0 0 ]};
rutas(6,2)={[102 82 85 83 76 64 54 48 49 34 27 0 0 0 0 0 0;...
        102 82 69 63 60 54 48 49 34 27 0 0 0 0 0 0 0;...
        106 120 137 121 104 82 85 83 76 64 54 48 49 34 27 0 0;...
        106 120 137 121 104 82 69 63 60 54 48 49 34 27 0 0 0;...
        106 117 107 88 83 76 64 54 48 49 34 27 0 0 0 0 0;...
        106 117 107 88 85 69 63 60 54 48 49 34 27 0 0 0 0]};
rutas(6,3)={[102 82 69 63 52 35 0 0 0 0 0 0 0;...
        106 120 137 121 104 82 69 63 52 35 0 0 0;...
        106 120 137 121 104 82 85 83 76 64 60 52 35;...
        102 82 85 83 76 64 60 52 35 0 0 0 0]};
```

```
rutas(6,4)={[102 103 71 0 0 0 0 0 0 0 0 0 0;...
        106 120 137 121 104 103 71 0 0 0 0 0 0 ;...
        106 117 107 88 85 82 103 71 0 0 0 0 0;...
        106 117 107 88 83 76 64 60 63 69 82 103 71]};
rutas(6,5)={[106 117 107 88 83 76 74 0 0 0;...
        106 117 107 88 85 69 63 60 64 74;...
        102 82 85 83 76 74 0 0 0 0;...
        102 82 69 63 60 64 74 0 0 0]};
rutas(6,6)={[]};
rutas(6,7)={[106 117 116 124 0 0 0 0 0 0 0 0 0;...
        106 117 107 88 83 81 108 124 0 0 0 0 0;...
        102 82 85 83 81 108 108 124 0 0 0 0 0;...
        102 82 69 63 60 64 76 81 108 124 0 0 0;...
        106 117 107 88 85 69 63 60 64 76 81 108 124]};
rutas(6,8)={[106 120 137 138 0 0;...
        102 104 121 138 0 0;...
        106 117 123 136 137 138]};
rutas(7,1)={[135 123 107 88 83 76 64 54 48 49 34 12 6 5 0 0 0;...
        135 136 137 121 104 82 85 83 76 64 54 48 49 34 12 6 5;...
        135 123 107 88 85 69 63 60 54 48 49 34 12 6 5 0 0;...
        135 136 137 121 104 82 69 63 60 54 48 49 34 12 6 5 0]};
rutas(7,2)={[135 123 107 88 83 76 64 54 48 49 34 27 0 0 0 0 0;...
        135 136 137 121 104 82 85 83 76 64 54 48 49 34 27 0 0;...
        135 123 107 88 85 69 63 60 54 48 49 34 27 0 0 0 0;...
        135 136 137 121 104 82 69 63 60 54 48 49 34 27 0 0 0]};
rutas(7,3)={[135 136 137 121 104 82 69 63 52 35 0 0 0;...
        135 123 107 88 85 69 63 52 35 0 0 0 0;...
        135 136 137 121 104 82 85 83 76 64 60 52 35;...
        135 123 107 88 83 76 64 60 52 35 0 0 0]};
rutas(7,4)={[135 136 137 121 104 103 71 0 0 0 0 0 0;...
        135 123 107 88 85 82 103 71 0 0 0 0 0;...
        135 123 107 88 83 76 64 60 63 69 82 103 71]};
rutas(7,5)={[135 123 107 88 83 76 74 0 0 0 0;...
        135 123 107 88 85 69 63 60 64 74 0;...
        135 136 137 121 104 82 85 83 76 74 0;...
        135 136 137 121 104 82 69 63 60 64 74]};
rutas(7,6)={[135 123 107 101 0 0 0 0 0 0 0 0 0 0;...
        135 136 137 121 104 82 85 88 101 0 0 0 0 0;...
        135 136 137 121 104 82 69 63 60 64 76 83 88 101]};
rutas(7,7)={[]};
rutas(7,8)={[135 136 137 138 0 0 0 0 0 0 0 0 0 0;...
        135 123 107 88 85 82 104 121 138 0 0 0 0 0;...
        135 123 107 88 83 76 64 60 63 69 82 104 121 138]};
rutas(8,1)={[122 119 104 82 69 63 60 54 48 49 34 12 6 5 0 0 0;...
```

```
        122 119 118 117 107 88 83 76 64 54 48 49 34 12 6 5 0;...
        122 119 104 82 85 83 76 64 54 48 49 34 12 6 5 0 0;...
        122 119 118 117 107 88 85 69 63 60 54 48 49 34 27 0 0]};
rutas(8,2)={[122 119 104 82 69 63 60 54 48 49 34 27 0 0 0 0 0;...
        122 119 118 117 107 88 83 76 64 54 48 49 34 27 0 0 0;...
        122 119 104 82 85 83 76 64 54 48 49 34 27 0 0 0 0;...
        122 119 118 117 107 88 85 69 63 60 54 48 49 34 27 0 0]};
rutas(8,3)={[122 119 104 82 69 63 52 35 0 0 0 0;...
        122 119 118 117 107 88 85 69 63 52 35 0;...
        122 119 104 82 85 83 76 64 60 52 35 0;...
        122 119 118 117 107 88 83 76 64 60 52 35]};
rutas(8,4)={[122 105 71 0 0;...
        122 119 104 103 71]};
rutas(8,5)={[122 119 104 82 85 83 76 74 0 0 0 0 0;...
        122 119 104 82 85 83 76 64 54 53 79 0 0;...
        122 119 118 117 107 88 83 76 74 0 0 0 0;...
        122 119 118 117 107 88 83 76 64 54 53 79 0;...
        122 119 104 82 69 63 60 64 74 0 0 0 0;...
        122 119 104 82 69 63 60 54 53 79 0 0 0;...
        122 119 118 117 107 88 85 69 63 60 64 74 0;...
        122 119 118 117 107 88 85 69 63 60 54 53 79]};
rutas(8,6)={[122 119 118 117 107 101 0 0 0 0 0 0;...
        122 119 104 82 85 88 101 0 0 0 0 0;...
        122 119 104 82 69 63 60 64 76 83 88 101;...
        122 119 104 82 69 63 60 64 76 81 100 101;...
        122 119 104 82 85 83 81 100 101 0 0 0]};
rutas(8,7)={[122 119 118 117 116 124 0 0 0 0 0 0 0 0 0;...
        122 119 104 82 85 83 81 108 124 0 0 0 0 0 0;...
        122 119 104 82 69 63 60 64 76 81 108 124 0 0 0;...
        122 119 118 117 107 88 83 81 108 124 0 0 0 0 0;...
        122 119 118 117 107 88 85 69 63 60 64 76 81 108 124]};
rutas(8,8)={[]};
```