UNIVERSIDAD AUTÓNOMA DEL
ESTADO DE MORELOS

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
CENTRO DE INVESTIGACIONES EN INGENIERÍA
Y CIENCIAS APLICADAS

ALGORITMO EVOLUTIVO PARALELIZADO PARA UNA CADENA DE SUMINISTRO CON VENTANAS DE TIEMPO

TESIS PROFESIONAL
PARA OBTENER EL GRADO DE:

DOCTORADO EN INGENIERÍA Y CIENCIAS APLICADAS
OPCIÓN TERMINAL EN TECNOLOGÍA ELÉCTRICA

PRESENTA:
OCOTLÁN DÍAZ PARRA

ASESOR INTERNO: Dr. MARCO ANTONIO CRUZ CHÁVEZ
ASESOR EXTERNO: Dra. MARÍA ELENA LÁRRAGA RAMÍREZ

CUERNAVACA, MOR. DICIEMBRE 2008

DEDICATORIAS

A mi amado esposo por su comprensión y entrega en tiempos difíciles; a mis padres y suegros por su ayuda moral y económica; a mis hijos Jorge y Fernando por su apoyo y aguante en el transcurso de este trabajo; a mi asesor por sus consejos, dirección y guía en la realización de este trabajo y a mi misma por mi persistencia ante este gran reto.

RESUMEN

El principal objetivo de esta tesis es el desarrollo de un algoritmo para el problema de ruteo vehicular con ventanas de tiempo, el cual deberá tener excelente costo/desempeño comparado con los mejores algoritmos existentes. El problema de ruteo vehicular con ventanas de tiempo es por su naturaleza un problema difícil de resolver. Para proponer una solución se debe tomar en cuenta la función objetivo y las variables implicadas en el problema así como las restricciones a las que están sometidas dichas variables. Dada la dureza del problema, las soluciones propuestas a este tipo de problemas difíciles son mediante heurísticas. En este trabajo de investigación la solución propuesta se compone de una heurística que combina un algoritmo genético con búsqueda en vecindades. Para lograr cumplir con el objetivo planteado se realizaron varias versiones del algoritmo hasta conseguir la mejor solución al problema de ruteo vehicular con ventanas de tiempo. Se realizó el análisis de sensibilidad consiguiendo los rangos de las variables que garantizan el mejor comportamiento del algoritmo. Se comparó la solución propuesta con otras soluciones de otros autores para medir el desempeño del algoritmo y ver si es una buena solución al problema de ruteo vehicular con ventanas de tiempo. Las pruebas realizadas se ejecutaron en el laboratorio de optimización y software y en la computadora Aleph de 32 procesadores. El análisis de resultados lleva a concluir que el algoritmo propuesto cumple con las expectativas planteadas en el objetivo. La estrategia de paralelización utilizada fue la de memoria compartida.

ABSTRACT

The main objective of this thesis is the development of an algorithm for the vehicle routing problem with time windows, which must have excellent cost/performance compared with the best existing algorithms. The vehicle routing problem with time windows is by its nature a problem difficult to solve. In order to offer a solution one is due to take into account the objective function and the variables implied in the problem as well as the restrictions which these variables are put under. Given the hardness of the problem, the propose solutions to this kind of difficult problems are by means of heuristic. In this work of investigation the propose solution is made up of a heuristic one that combines a genetic algorithm with search in vicinities. In order to manage to fulfill the raised objective several versions of the algorithm were realized until securing the best solution to the vehicle routing problem with time windows. The sensitivity analysis was realized obtaining the ranks of the variables that guarantee the best behavior of the algorithm. The solution propose with other solutions of other authors was compared to measure the performance of the algorithm and to see if it is a good solution to the vehicle routing problem with time windows. The realized tests were executed in the laboratory of optimization and software and in the Aleph computer of 32 processors. The analysis of results takes to conclude that the proposed algorithm fulfills the expectations raised in the objective. The strategy of used paralelization was the one of shared memory.

AGRADECIMIENTOS

A CONACYT por haberme brindado el apoyo económico sin el cual no habría podido realizar mis estudios de doctorado.

A los doctores directores, miembros del comité tutoral y revisores de tesis: Marco Antonio Cruz Chávez, María Elena Lárraga Ramírez, José Crispín Zavala Díaz, David Juárez Romero, José Alfredo Hernández Pérez, Margarita Tecpoyotl Torres y Álvaro Zamudio Lara, por sus acertados comentarios al trabajo de investigación para mi crecimiento intelectual.

A la Universidad autónoma del Estado de Morelos por las facilidades prestadas en el uso de la supercomputadora Aleph de 32 procesadores IBM-p690 a través del proyecto “Cómputo científico”. FOMES2000-SEP.

Al laboratorio de computo del Centro de Investigaciones en Ingeniería y Ciencias Aplicadas por el apoyo en la etapa final de la realización de pruebas.

A fundación TELMEX por brindarme apoyo para material didáctico y conexión a internet

ÍNDICE

LISTA DE TABLAS

LISTA DE FIGURAS

Capítulo 1. Introducción

Uno de los problemas a los que diariamente nos enfrentamos dentro de una sociedad, es la manera de cómo transportarnos para llegar a nuestras actividades diarias puntualmente. El problema del transporte engloba la mayor parte del entorno global en la sociedad ya que forma una parte importante en la cadena de suministros. El comportamiento de problemas del vivir diario se pueden representar por medio de las matemáticas, en su rama de optimización combinatoria. La optimización combinatoria estudia problemas difíciles de resolver [Papadimitriou y Steiglitz, 1998]. Un problema de optimización combinatoria tiene la característica de presentar dificultad para encontrar una buena solución, a la vez que es imposible enumerar explícitamente todas las soluciones posibles de las cuales sólo una es la óptima. Un problema combinatorio es un modelo matemático con un sistema de ecuaciones y expresiones matemáticas relacionadas que describen la particularidad del problema, cada modelo presenta una función objetivo y un conjunto de restricciones [Hillier y Lieberman, 1991].

Un problema de optimización combinatoria se caracteriza por contar con un modelo que consta de un conjunto de variables que deben maximizar o minimizar una función objetivo, sometida a un conjunto de restricciones [Martí, 2003]. Los problemas de optimización combinatoria existentes en la literatura se clasifican en los siguientes conjuntos: P,NP y NP-Hard; P es la clase de lenguajes reconocibles en tiempo polinomial por el modelo de turing de máquinas deterministas [Karp, 1972], NP es la clase de lenguajes reconocibles por el modelo de turing de máquinas no-deterministas [Karp, 1972], NP-Hard es un tipo de problema que debe cumplir con la condición de que cada problema en NP tiene que ser reducible a él [Cook, 1971]. Los problemas clásicos comúnmente abordados por optimización combinatoria

son el problema de la mochila, el problema de bin packing, el problema de satisfactibilidad, el problema del viajero, el problema de asignación de horarios, el problema de job shop y el problema del transporte entre otros.

En el presente trabajo se realiza un análisis del problema del transporte con tiempos de atención y su solución mediante métodos de optimización combinatoria. Los métodos de optimización combinatoria para búsqueda de soluciones a los problemas de optimización anteriormente mencionados, se presentan en la siguiente sección.

1.1. Algoritmos de optimización combinatoria

Resolver un problema de optimización consiste en encontrar el valor que deben tomar las variables para hacer óptima la función objetivo satisfaciendo el conjunto de restricciones para una instancia dada. Una instancia de un problema es obtenido mediante la especificación de valores particulares sobre los parámetros de un problema [Garey y Johnson, 1979]. La solución a un problema combinatorio se construye por medio de métodos heurísticos cuando las instancias de entrada son grandes. Un método heurístico es un procedimiento para resolver un problema matemático bien definido mediante aproximaciones en un tiempo de computación razonable sin garantizar la optimalidad [Whitley, 1993].

Para aproximarse a una solución un tipo de algoritmo que utilizan los métodos heurísticos es el algoritmo no determinístico. Un algoritmo no determinístico es aquel que provee diferentes soluciones en un tiempo de ejecución dado [Garey y Johnson, 1979]. Las heurísticas más conocidas en literatura son algoritmos genéticos, recocido simulado, búsqueda tabú, colonia de hormigas, búsqueda dispersa entre otros. En optimización

combinatoria básicamente hay que tener presente dos aspectos, la eficacia y la eficiencia del algoritmo. La eficacia está representada por las soluciones acertadas del algoritmo propuesto a un problema combinatorio [Papadimitriou y Steiglitz, 1998]. La eficiencia está representada por el tiempo que tarda el algoritmo propuesto para problemas combinatorios en encontrar una solución [Papadimitriou y Steiglitz, 1998]. Para entender sobre la eficacia y eficiencia de un algoritmo, es necesario entender cómo afectan algunos factores como es, el tamaño de la entrada, el equipo utilizado para la ejecución del algoritmo y la complejidad del algoritmo. En la sección siguiente se explica cómo calcular la complejidad de un algoritmo y los tipos de complejidad que existen.

1.2. Complejidad de los algoritmos

La teoría de la complejidad computacional está relacionada con la medición de la dificultad de los cálculos computables. La medición muestra cuantos pasos toma evaluar cualquier algoritmo sobre cualquier argumento [Hartmanis y Hopcroft, 1971]. Los recursos comúnmente estudiados son el espacio (cantidad de memoria utilizada para resolver un problema) y el tiempo (número de pasos de ejecución de un algoritmo para resolver un problema). Un algoritmo que resuelve un problema pero que tarda mucho en hacerlo, difícilmente será de utilidad. Igualmente un algoritmo que necesite gran cantidad de memoria no será probablemente utilizado. Si un cálculo en un algoritmo requiere más tiempo que en otro decimos que éste es más complejo y que cuenta con complejidad temporal mayor. Por otro lado, si un cálculo requiere más espacio de almacenamiento que otro decimos que es más complejo, en este caso hablamos de una complejidad espacial.

Un algoritmo es visto como una caja negra que necesita datos de entrada y un proceso donde se realizan los cálculos [Pressman, 2006]. A los datos de entrada al problema se les conoce como tamaño de la entrada o número de datos a procesar por el algoritmo, dependiendo del problema que se esté utilizando, entre mayor sea el número de datos a procesar mayor será el tiempo requerido para la realización de los cálculos. Al tiempo requerido para los cálculos se le conoce como tiempo de ejecución. La función temporal que representa un algoritmo está en función del tamaño de la entrada del problema y ésta función da la complejidad temporal del algoritmo en base al número de instrucciones que requiere el algoritmo.

Por ejemplo digamos que el tamaño de la entrada es *n* y que el tiempo de ejecución promedio en base al tamaño de la entrada es *T(n)*. Independientemente del tipo de problema que se esté analizando siempre existe un mejor caso y un peor caso; para esta consideración se toma un intervalo de *T(n)* entre un mínimo y un máximo de tal forma que $T_{min}(n)<T(n)<T_{max}(n)$. Siendo el extremo $T_{max}(n)$ el peor caso. El análisis de eficiencia se lleva a cabo estudiando los algoritmos en casos extremos. Por ejemplo cuando *n* tiende a infinito se dice que tiene un comportamiento asintótico; digamos que *g(n)* es un conjunto de funciones que determinan el uso de recursos, las funciones *g* serán clasificadas tomando como criterio de agrupación su comportamiento asintótico. Un grupo de funciones *g* con un mismo comportamiento asintótico será llamada un orden de complejidad, estas familias se designan con la notación *O ()*. Cada conjunto *g* tiene un representante de familia representado por *f(n)*, de tal forma que el conjunto de funciones *g* que son del orden *f(n)* se denota como: $g \supset O(f(n))$.

El conjunto de *O(f(n))* es el de las funciones de orden *g(n)* que se define como:

$$O(f(n)) = \{g \mid \exists k \in \Re^+, \exists n_0 \in \aleph, \forall n \geq n_0 : g(n) \leq kf(n)\}$$

Donde *O(f(n))* está formado por aquellas funciones *g(n)* que crecen a un ritmo menor o igual que el de *f(n)*. Las funciones *g* que forman este conjunto *O(f(n))* se dice que están dominadas asintóticamente por *f*, en el sentido de que para n suficientemente grande y salvo una constante multiplicativa *k*, *f(n)* es una cota superior de *g(n)*. Y *k* es la constante multiplicativa (que permite agrupar funciones como n^2 y $5n^2$) y n_0 indica a partir de qué punto *f* es realmente cota superior de *g*.

Por ejemplo se puede comprobar que la función *g(n)*= $3n^3+2n^2$, es de orden de complejidad $O(n^3)$. Aplicando la definición dada se tiene que: *g(n)*= $3n^3+2n^2$, $f(n)=n^3$, $n_0=0$, con k=5 para n=1, por lo que $3n^3+2n^2 \leq 5n^3$. Para reafirmar dicha definición se probaron diferentes valores de *n* presentados en la Tabla 1.1.

Tabla 1.1. Evaluación de la función g(n) y f(n)

N	g(n)	kf(n)
100	3020000	5000000
2000	2.4008E+10	4E+10
40000	1.92E+14	3.2E+14
800000	1.536E+18	2.56E+18
1000000	3E+18	5E+18
10000000	3E+24	5E+24
100000000	3E+27	5E+27

Como se observa en la Tabla 1.1 el tiempo de ejecución de *g(n)* es proporcional al tiempo de ejecución de *kf(n).* De tal forma que un algoritmo cuyo tiempo de ejecución, para *n* suficientemente grandes, sea $T=3n^2+6n$ se puede considerar proporcional a n^2 en este caso se puede asegurar que el algoritmo es del orden de n^2 y se escribe $O(n^2)$. La notación *O ()* representa el orden de complejidad de un algoritmo en el peor caso. En la Tabla 1.2 se presentan las funciones de complejidad algorítmica más habituales,

ordenadas de mayor a menor eficiencia, en las cuales el único factor del que dependen es del tamaño de la entrada *n*.

Tabla 1.2. Órdenes de complejidad

Representación	Descripción
O(1)	Orden constante
O(log n)	Orden logarítmico
O(n)	Orden lineal
O(n log n)	Orden cuasi-lineal
$O(n^2)$	Orden cuadrático
$O(n^3)$	Orden cúbico
$O(n^a)$	Orden polinómico
$O(2^n)$	Orden exponencial
O(n!)	Orden factorial

En la figura 1.1 se presenta el comportamiento con respecto al tiempo de las funciones de grado: $n\log n$, n^3, $n^{\log n}$ y $n!$. A medida que crece el tamaño de *n,* que representa una de las variables de la instancia, crece el tiempo de ejecución que depende de la complejidad del algoritmo, de aquí la importancia de construir un algoritmo con complejidad polinómica a lo más de grado 3, y lo mejor, un algoritmo con complejidad logarítmica para garantizar la eficiencia. Un algoritmo con complejidad exponencial difícilmente hará computable al VRPTW y no garantizará que la solución sea eficiente y por lo tanto no será útil para un análisis combinatorio porque lo que se busca en optimización combinatoria es que los algoritmos sean eficientes y eficaces.

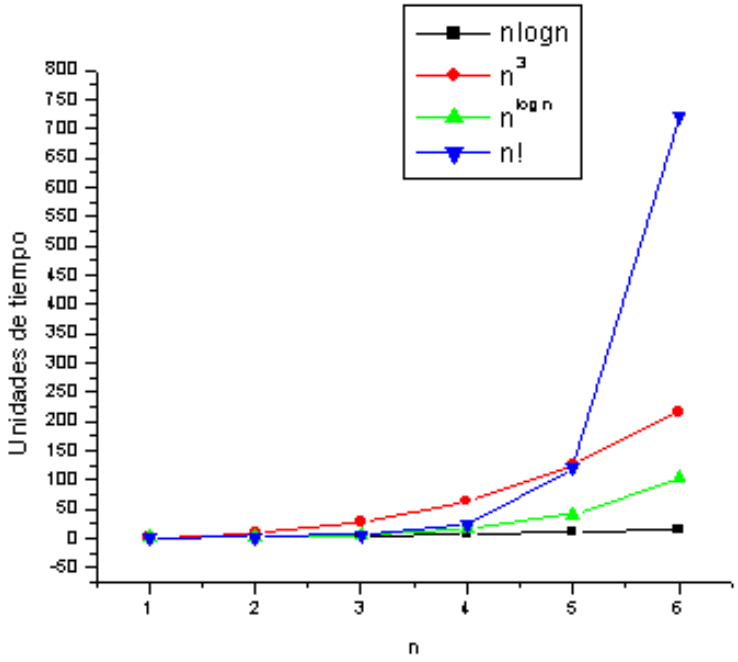

Figura 1.1. Comportamiento de las funciones con respecto al tiempo

La importancia de diseñar un algoritmo con orden de complejidad logarítmica o polinomial, es que independientemente de la velocidad del equipo en el que se está ejecutando, el tiempo de ejecución del algoritmo será aceptable. Por ejemplo Contamos con una computadora capaz de procesar datos en 10^{-4} segundos. En esta computadora se ejecuta un algoritmo que lee registros de una base de datos, dicho algoritmo tiene una complejidad exponencial 2^n, ¿Cuánto tiempo se tardará en procesar una entrada de *n* datos? Los cálculos se presentan en la Tabla 1.3.

Tabla 1.3. Tiempos del algoritmo en base al tamaño de la entrada n para un algoritmo con complejidad exponencial 2^n

Entrada n	TIEMPO
10	≈ 1 décima de segundo
20	≈ 2 minutos
30	> 1 día
40	> 3 años
50	= 3 570 años
100	= 4,019,693,684,133,150 milenios

Ahora se tiene la misma computadora capaz de procesar datos en 10^{-4} segundos. Pero se ejecuta un algoritmo que hace el mismo trabajo antes citado, pero este algoritmo tiene una complejidad cúbica n^3, ¿Cuánto tiempo tardará en procesar una entrada de *n* datos? Los cálculos se presentan en la Tabla 1.4.

Tabla 1.4. Tiempos de ejecución en base al tamaño de la entrada con un algoritmo con complejidad n^3

Entrada n	TIEMPO
10	= 1 décima de segundo
20	= 8 décimas de segundo
100	= 1.7 minutos
200	= 13.3 minutos
1000	≈ 1 día

Se puede concluir, que sólo un algoritmo eficiente, con un orden de complejidad bajo, puede tratar grandes volumen de datos [Joyanes, 2000].

Entonces un algoritmo es:

Muy eficiente si su complejidad es de orden *log n*

Eficiente si su complejidad es de orden n^a

Ineficiente si su complejidad es de orden 2^n

Se considera que un problema es tratable si existe un algoritmo que lo resuelve con complejidad menor que 2^n, y que es intratable en caso contrario [Garey y Johnson, 1979]. Entonces una complejidad deseable para un algoritmo de optimización combinatoria es de orden logarítmico.

Para construir el polinomio *g(n)* y calcular la complejidad del algoritmo se parte del pseudocódigo y se aplican las reglas y estimaciones definidas en el apéndice A de este documento.

Conociendo como obtener la complejidad del algoritmo es deseable saber que tipos de algoritmos heurísticos se pueden construir para problemas intratables con complejidad polinomial. En la siguiente sección se habla de las principales heurísticas como métodos de solución propuestos para problemas de optimización.

1.3. Principales métodos de solución a problemas NP

Un problema NP es una clase de problema de optimización difícil de resolver, porque no se puede garantizar encontrar la mejor solución posible en un tiempo razonable y por su naturaleza requiere ser resuelto mediante una heurística. Las heurísticas comúnmente utilizadas en optimización combinatoria [Díaz *et al.,* 1996] [García, 2005] como alternativa de solución al tipo de problemas NP son: algoritmos genéticos, recocido simulado, búsqueda tabú, procedimientos de búsqueda adaptable utilizando un algoritmo tipo glotón aleatorio llamado GRASP (por sus siglas en inglés) entre otros.

Los algoritmos genéticos son una clase de algoritmos inspirados en los mecanismos de la genética y de la teoría de la evolución de Darwing, que

se aplican a problemas de optimización, según Holland un algoritmo genético es un procedimiento basado en la analogía con la evolución de los seres vivos. La premisa que subyace a este tipo de enfoques es, que se puede encontrar soluciones aproximadas a problemas de gran complejidad computacional mediante un procedimiento de evolución simulada matemáticamente en un ordenador [Holland, 1975]. Este tipo de algoritmo permite a partir de una población inicial de soluciones, obtener soluciones potencialmente mejores mediante el cruce de varias de las soluciones originales. Se requiere del diseño de tres operadores, selección, cruzamiento y mutación.

El recocido simulado es un algoritmo basado en el recocido de sólidos, el modo de trabajo del recocido es buscar un estado de mínima energía enfriando poco a poco el material [Kirkpatrick et al., 1983]. Como método heurístico permite movimientos que no sean de mejora, la probabilidad de aceptación de movimientos malos disminuye con el tiempo.

La búsqueda tabú es un algoritmo que se basa en la gestión de memoria sobre los espacios explorados; donde se prohíbe al sistema deshacer algunos de los últimos movimientos realizados, para permitir la búsqueda por entornos que de otro modo no se explorarían [Glover y Laguna, 1997].

El algoritmo GRASP es un algoritmo que empieza buscando la solución del problema en diferentes puntos del espacio de soluciones y una vez que encuentra una solución utiliza un procedimiento de búsqueda local para mejorar ésta solución alcanzando un óptimo local.

En este trabajo de investigación se pretende trabajar con este tipo de heurísticas en específico con algoritmos genéticos aplicados al problema de ruteo vehicular y logística como se describe en la siguiente sección.

1.4. Problemática de la investigación e hipótesis

El transporte es un área de decisiones clave en la logística, actualmente absorbe un 35% de los costos de un producto, en lo que implica los fletes y movimientos [Balseiro, 2007]. Un ahorro en costo por producto por muy mínimo que sea a gran escala es un ahorro en costo considerable para la empresa. Por esta necesidad creciente de ahorro en costos de transportación y logística surge la inquietud de desarrollar una alternativa de solución para las empresas. La problemática del transporte y logística no es nueva. A finales de los 50's se comenzaron a estudiar académicamente los problemas de transporte y distribución para convertirse luego en uno de los más importantes dentro del área de investigación operativa, principalmente por el hecho de aparecer en una gran cantidad de situaciones prácticas, por su complejidad y dificultad de solución. Por su naturaleza, el problema de ruteo vehicular pertenece a una clase computacionalmente difícil (NP-hard) [Toth y Vigo, 2002] [Lenstra y Rinnoy, 1981]. Aún con la creciente capacidad de cálculo de las computadoras los algoritmos que buscan soluciones exactas son de poca utilidad en la práctica ya que requieren una cantidad de tiempo que crece exponencialmente conforme aumenta el tamaño del problema [Garey y Johnson, 1979]. Surge entonces la necesidad de heurísticas que proporcionen un resultado aproximado, esto es, se busca una buena solución aunque no sea necesariamente la mejor. Consecuente con el desarrollo académico en las últimas décadas ha comenzando a aumentar la utilización de software de optimización para la distribución de bienes y servicios en la industria. Según Toth y Vigo [Toth y Vigo, 2002] las aplicaciones en casos reales han demostrado que el ahorro logrado por estas técnicas en los procesos de distribución es substancial: del 5% al 20%. Teniendo en cuenta la importancia de los costos de logística en el costo total

del producto se puede ver que el impacto de estos ahorros es muy importante para la economía.

El ruteo y programación de vehículos computarizados (RPVC) proporciona múltiples beneficios; por ejemplo el software puede ser utilizado como una herramienta estratégico-táctica que permite buscar respuestas ante distintos escenarios posibles. Se pueden realizar simulaciones que modifiquen algún parámetro de la distribución y observar el impacto de la decisión en término de costos o calidad de servicio. Los resultados obtenidos son considerablemente más precisos y de mayor validez que aquellos que los conseguidos por algún método manual.

Existe entonces una oportunidad de poder participar en la aportación de ideas y cuestionamientos para la construcción de soluciones para el problema del transporte y logística. En esta investigación se plantea la siguiente hipótesis:

“¿Es posible construir una solución al problema del transporte y logística combinando métodos, técnicas y herramientas dentro del ámbito computacional que den una solución acertada en tiempo polinomial?”

1.5. Motivación de la tesis

La inquietud de la mayoría de los empresarios, es la minimización de gastos y maximización de ganancias. Lo que implica que se busque mejorar en todos los aspectos de la industria en general, especialmente en aquellos aspectos que consumen mayor ingreso como lo es la logística en el transporte. Un ahorro en el transporte por mínimo que sea es considerado ganancia para la empresa. Normalmente en la industria cuando se habla de

logística y transporte se ve involucrado el tiempo, (un factor muy importante para la producción). Para ilustrar la importancia del tiempo en el transporte supongamos que una *empresa-X* (proveedor de Y) debe entregar a una *empresa-Y* un pedido de 10,000 unidades de piezas de carrocería para ensamblado en tres días y la *empresa-Y* debe entregar los autos ensamblados en 7 días. La *empresa-Y* cumplirá siempre y cuando tenga el material para ensamblaje a tiempo, un retraso en la entrega de la *empresa-X*, afectará en gran medida la producción de la *empresa Y*. Un mal cálculo en la asignación de rutas y atención de tiempos provocará serios problemas de costo, que redunda en pérdidas económicas para la empresa-Y. Considerando la oportunidad de ofrecer a la industria de transporte y logística otra alternativa de ahorro en gastos de transportación, surge la inquietud de desarrollar un algoritmo de calendarización de la mejor ruta implicando tiempos de atención y menor costo. El problema tratado en literatura que por sus condiciones se acerca más a las condiciones cotidianamente requeridas en la logística es el conocido “problema de ruteo vehicular con ventanas de tiempo” que es el que de ahora en adelante será referenciado como VRPTW (por sus siglas en inglés que significan Vehicle Routing Problem With Time Windows). El algoritmo de calendarización de rutas se probará para instancias en literatura conocidas como benchmarks, específicamente se usarán los benchmarks de Solomon que fueron especialmente definidos para el problema de ruteo vehicular con ventanas de tiempo. El algoritmo de calendarización de rutas se implementará por medio de una heurística paralelizada, combinando diferentes técnicas que se especificarán en los capítulos subsiguientes de ésta tesis.

1.6. Objetivos de la tesis

En busca de la solución al problema del transporte y logística involucrando tiempos de atención se plantean los siguientes objetivos.

Objetivo general:

Desarrollo de un algoritmo híbrido paralelo para el problema de ruteo vehicular con ventanas de tiempo, el cual deberá ser competitivo en costo/desempeño comparado con los mejores algoritmos existentes.

Objetivos específicos.

-Desarrollo de un algoritmo con complejidad polinomial para un modelo teórico del problema de ruteo vehicular con ventanas de tiempo.

-Análisis y comprobación de la eficiencia y eficacia del algoritmo utilizando benchmark existentes.

1.7. Alcances y limitaciones

Dentro de los alcances de la tesis esta la obtención de un algoritmo con procesamiento en paralelo que sea eficiente y eficaz en la solución del problema de ruteo vehicular con ventanas de tiempo, para un modelo teórico definido probado para instancias grandes reportadas en la literatura para este problema.

La implementación del algoritmo para instancias reales esta fuera de los alcances de este documento pero está considerado como trabajo futuro.

1.8. Contribuciones

La principal aportación de este trabajo de tesis es la construcción de una alternativa de solución, que sea eficaz y eficiente, para problemas de optimización combinatoria, especialmente aplicados al problema de ruteo vehicular con ventanas de tiempo y probada experimentalmente con instancias grandes. El algoritmo probado experimentalmente se pretende aplicar en un futuro a la industria del transporte y logística con instancias reales y contribuir al ahorro de recursos comúnmente gastados en la transportación. Actualmente la transportación es sus diferentes ramas se considera como un área prioritaria para nuestro país [Conacyt, 2008]. El Consejo Nacional de Ciencia y tecnología ha financiado proyectos a nivel nacional e internacional que giran en torno al tema del transporte en sus diferentes rubros [Conacyt, 2008 b], lo que deja clara la oportunidad que existe de participar activamente con esta propuesta de solución al problema de transporte presentada en este trabajo de investigación y contribuir al desarrollo del país.

Capítulo 2. El problema del transporte

El problema del transporte ha sido de vital consideración en el ámbito social y económico. Independientemente del medio que se utilice para transportar ya sean productos o personas y del crecimiento de la tecnología; el transporte sigue siendo tema de consideración mundial. Con el propósito de ofrecer una alternativa de solución a tan grande problemática se ha desarrollado esta investigación en torno al problema del transporte.

2.1. Descripción del problema del transporte y su impacto socioeconómico

El problema del transporte en literatura se conoce como VRP (por sus siglas en inglés, Vehicle Routing Problem), y consiste en *determinar un conjunto de rutas* para una flota de vehículos que parten de uno o más depósitos o almacenes para satisfacer la demanda de varios clientes dispersos geográficamente [Dantzing y Ramser, 1959], [Toth y Vigo, 2001]. El objetivo de estudio de este problema es satisfacer la demanda de dichos clientes *minimizando el costo total* de recorrer las rutas. Se cuenta con una flota de vehículos que parte del depósito, el depósito es el lugar de donde parten y regresan los vehículos una vez que han concluido su recorrido. Un recorrido o ruta, implica la asignación de clientes por turno a un vehículo, la asignación de clientes al vehículo dependerá de la demanda del cliente y de la capacidad del vehículo. Si un vehículo ha llegado a su límite de capacidad los siguientes clientes serán asignados a otro vehículo hasta cumplir su capacidad y así sucesivamente hasta satisfacer a todos los clientes. Una vez asignados los clientes a sus respectivos vehículos, el conjunto de todas las rutas o recorridos de cada uno de los vehículos

implicados constituye la ruta total, conocida como una solución al problema del transporte. Para darnos una idea del funcionamiento del problema VRP, en la Figura 2.1 se muestra la representación de una posible solución. En el ejemplo se cuenta con una flota de 4 vehículos para atender a 10 clientes distribuidos en el plano (x,y). La ruta solución es el conjunto de todas las rutas recorridas por los cuatro vehículos (A, B, C y D) los cuales atienden a diferentes clientes en cada ruta, por ejemplo el vehículo A atiende al cliente 8, el vehículo B atiende a los clientes 2, 1 y 7, el vehículo C atiende a los clientes 3, 4, y 10, y el vehículo D atiende a los clientes 5, 6, y 9.

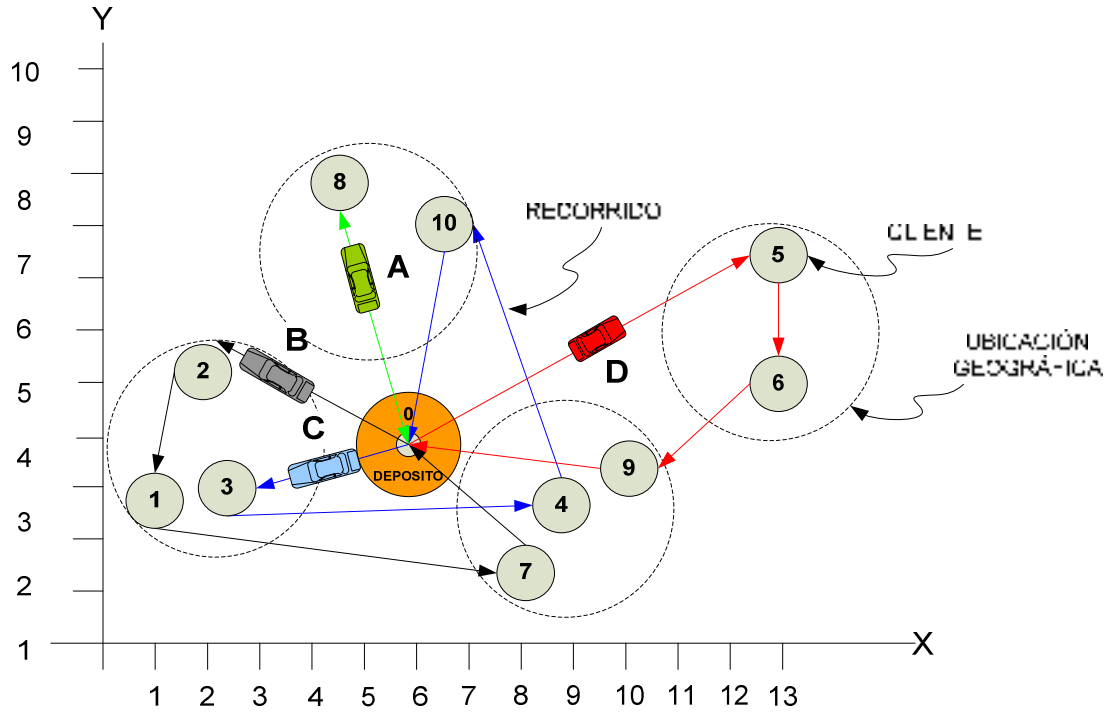

Figura 2.1. Representación de una solución al problema VRP

El VRP aparece de forma natural en las áreas de transporte, distribución y logística [Dantzing y Ramser, 1959]. La logística es el proceso que se encarga de planear, implementar y controlar eficaz y efectivamente el transporte, incluyendo el almacenamiento de bienes o servicios y la

información entre un origen y un destino. La misión de la logística es contar con los bienes o servicios necesarios, en el lugar, tiempo y condición deseada, previendo las contingencias y administrando los costos. Una cadena logística, es el conjunto de eslabones necesarios para satisfacer el posicionamiento de bienes o servicios, bajo características de eficacia y eficiencia, en abastecimiento, producción y distribución. La distribución constituye el mecanismo de enlace entre el abastecimiento y la producción, y cierra el conjunto estructural de la cadena logística, es aquí donde el transporte se ve principalmente involucrado con la economía y la sociedad.

En muchos sectores del mercado, *el transporte implica un gran costo* que se carga a los productos que se distribuyen. La utilización de métodos de computación para calcular las rutas óptimas o cercanas a las óptimas puede suponer unos ahorros de costos importantes (del orden del 5% al 20%, [Garey y Johnson, 2003]). De forma global las infraestructuras de transporte son un factor imprescindible para el desarrollo socioeconómico de una nación. Según la OCDE (Organización para la Cooperación y el Desarrollo Económico) la participación del transporte en el PIB de Francia fue del 5%, Alemania 6%, Italia 7% y U.S.A. 6% en el año de 1984, es decir hace ya 20 años estos países tenían claro que las Infraestructuras de Transporte en una Nación son Fundamentales para el Desarrollo Económico. En México el Sector de Transportes ocupa el tercer lugar como las principales actividades económicas del país generadoras de Valor Agregado Bruto con una tasa de crecimiento en el período 1993-2001 de 3.1% anual según el Instituto Mexicano del Transporte [Balbuena y Herrera, 2003]. [Antún y Briceño, 2001]

Debido a la importancia que tiene el problema del transporte con la economía y la sociedad, es de interés para la comunidad científica aportar alternativas de solución a problemas reales del transporte, sin embargo, esta tarea puede ser sumamente difícil por todas las variables implicadas en el

problema de transporte, por lo que en literatura se ha seccionado el problema del transporte para poder resolver de manera específica cada una de las partes que forman el problema del transporte en general.

2.2. Variantes del problema del transporte

En los problemas reales de VRP existen muchas *restricciones,* entre las que cabe citar: que cada vehículo tenga una *capacidad limitada,* que cada cliente tenga que ser atendido dentro de un margen de tiempo (*ventana de tiempo*), que los puntos de suministro sean varios, que los clientes deban ser atendidos por varios vehículos, que las entregas se hayan de realizar en determinados días. Todas estas consideraciones hacen que el VRP sea difícil de implementar en problemas reales, por lo que se le ha seccionado en problemas que consideran sólo algunas restricciones de manera específica; a estos problemas se les conoce como variantes de VRP [Toth y Vigo, 2001] [*Gonzalez y Aristizabal, 2006*]. Las variantes más comunes en VRP son las siguientes:

1.- MDVRP con múltiples depósitos (por sus siglas en inglés, Multi-Depot Vehicle Routing Problem) [*Tansini et al., 2008*]. Una empresa puede disponer de varios depósitos o almacenes desde los que suministra la demanda de sus clientes. Los clientes y los depósitos están mezclados, entonces para resolver un problema con estas características se necesita asignar los clientes a los depósitos. Para cada depósito se tiene una flota de vehículos. Cada vehículo que parte de un depósito, sirve a los clientes asignados a ese depósito y después regresa a dicho depósito. El objetivo del problema es servir a todos los clientes minimizando el número de vehículos

y la distancia total viajada. Un ejemplo de este problema es la red repartidora de pizas

2.- PVRP periódico (por sus siglas en inglés, Period Vehicle Routing Problem) [Francis et al., 2004]. En el problema VRP clásico, el periodo de planificación es un día. En el caso del problema PVRP, el periodo de planificación se extiende a M días. El objetivo es minimizar la flota de vehículos y el tiempo total de viaje. Un vehículo puede no regresar al depósito el mismo día de su partida. Durante el periodo de M días, cada cliente debe ser visitado al menos una vez. Un ejemplo de este problema son las exportaciones e importaciones de productos mexicanos y extranjeros.

3.- SDVRP de entrega dividida (por sus siglas en inglés, Split Delivery Vehicle Routing Problem) [Lee et al., 2002] [Archetti et al., 2001]. Se trata de un problema VRP en el que se permite que un cliente pueda ser atendido por varios vehículos si el coste total se reduce. Esto es importante si el tamaño de los pedidos de un cliente excede la capacidad de un vehículo. El objetivo es minimizar la flota de vehículos y el tiempo total de viaje. Un ejemplo de este problema es el suministro de materia prima de una industria de la construcción.

4.- SVRP estocástico (por sus siglas en inglés, Stochastic Vehicle Routing Problem) [Bianchi, 2005]. Se trata de un VRP en que uno o varios componentes son aleatorios. Por ejemplo: Clientes aleatorios. Un cliente aleatorio *i*, es un cliente que tiene una probabilidad p_i de estar presente y una probabilidad *1-p* de estar ausente. Demandas estocásticas: la demanda del cliente *i*, representada por *di*, tiene una determinada distribución de probabilidad. Tiempos estocásticos: el tiempo de servicio, t_i, y los tiempos de viaje t_{ij}, son variables aleatorias. Cuando algunos datos son aleatorios no es posible cumplir con todas las restricciones. Por tanto se puede llevar a cabo

ciertas acciones correctivas cuando una restricción es violada. Un ejemplo de este problema es el sistema de rutas de ambulancias en un hospital.

5.- VRPPD con recolección y entregas (por sus siglas en inglés, Vehicle Routing Problem Pickup and Delivery) [Volkan, 2005] [Dethloff, 2002] [Halse, 1992] [Gendreau et al., 1994] [Min, 1989]. Es un VRP en que cabe la posibilidad de que los clientes pueden devolver determinados bienes. Por tanto, se debe tener presente que los bienes devueltos por los clientes caben en el vehículo. Esta restricción hace más difícil el problema de planificación y puede obligar a una mala utilización de las capacidades de los vehículos, un aumento de las distancias recorridas y un mayor número de vehículos. Por todo lo dicho, se suelen considerar situaciones tales como recorridos de ida y vuelta, que en el recorrido de ida se realicen las entregas y en el recorrido de vuelta al depósito se hagan las devoluciones, de manera que no hay intercambio de bienes entre clientes. Otra alternativa es relajar la restricción de que todos los clientes deben ser visitados exactamente una vez. El objetivo es minimizar la flota de vehículos y el tiempo total de recorrido con la consideración de que el vehículo debe tener suficiente capacidad para transportar los bienes a entregar así como los devueltos para regresarlos al depósito. Un ejemplo de este problema es sistema de carritos repartidores de la línea BIMBO y Marinela a tienditas de la esquina.

6.- VRPB con devoluciones (por sus siglas en inglés, Vehicle Routing Problem with Backhauls) [Charlotte y Goetschalckx, 1998]. El VRPB es un VRP en que los clientes pueden demandar o devolver artículos. Por tanto se necesita tener en cuenta que los bienes que los clientes devuelven caben en el vehículo. Pero además, se debe cumplir que todas las entregas se realizan antes de las devoluciones. Esto se debe al hecho de que los vehículos se cargan por la parte trasera y que la recolocación de la carga en los vehículos se considera antieconómica o no factible. Las cantidades demandadas y las devueltas se conocen de antemano. El VRPB es similar al

VRPPD con la restricción de que en el caso del VRPB todas las entregas de una ruta se deben completar antes de las devoluciones. El objetivo es encontrar un conjunto de rutas que minimiza la distancia total recorrida. Un ejemplo de este problema es el sistema repartidor de productos a grandes almacenes como Comercial Mexicana y Walmart entre otros.

7.- VRPTW con ventanas de tiempo (por sus siglas en inglés, Vehicle Routing Problem with Time Windows) [Kohl et al., 1997] [Toth y Vigo, 2001] [Solomon, 1985]. Es un VRP con la restricción adicional de que se asocia una ventana de tiempo con cada cliente. Al cliente *i*, se le asocia la ventana de tiempo *[ei, li]*. Si un vehículo llega al cliente antes del instante *ei* el vehículo espera hasta ese instante para ser atendido por el cliente. Si llega en el intervalo de la ventana de tiempo, el vehículo suministra la demanda en el momento de la llegada y finalmente si el vehículo llega con posterioridad a *li* entonces el cliente no atenderá al chofer del vehículo. El objetivo es minimizar la flota de vehículos, el tiempo total de viaje así como el tiempo total de espera al suministro de los clientes. Un ejemplo de este problema es el transporte escolar, la calendarización del transporte urbano, el transporte aéreo, el transporte naviero entre otros.

Todas las variantes arriba descritas tienen una aplicación en los problemas reales de transportación. La variante de VRP que se analizará en este trabajo de investigación es la variante VRPTW. Esta variante define un problema cuya principal característica es el uso de tiempos de atención al cliente. En la siguiente sección se describe a detalle el problema VRPTW.

2.3. Descripción del problema del transporte con ventanas de tiempo

El problema de ruteo de vehículos con ventanas de tiempo tiene diferentes áreas de aplicación en logística, investigación operativa, transporte, distribución, área de economía aplicada y transporte escolar, por citar sólo algunos, debido a que el problema radica en minimizar los costos de la transportación sujeta a restricciones de tiempo y de capacidad en base a la demanda de cada cliente.

El problema del transporte con ventanas de tiempo contiene restricciones que deben cumplirse para poder construir una solución. Tales restricciones son: el número de vehículos con que cuenta la empresa, la capacidad de cada vehículo, el número de clientes a atender, demanda de cada cliente y el tiempo de atención que demanda cada cliente. A esta última restricción se le conoce como ventana de tiempo, que es lo que caracteriza principalmente a esta variante. Una ventana de tiempo es un intervalo en el cual un cliente desea ser atendido por un vehículo en una ruta. Cuando a las variables de un modelo se le asignan ciertos valores a éste modelo se le conoce como una instancia del problema al cual se le puede buscar una solución. En literatura existe la representación de instancias para el problema de ruteo de vehículos con ventanas de tiempo propuestas por Solomon [Solomon, 1987]. Las instancias están clasificadas por tipo y por clase, se tiene dos tipos; el tipo 1 maneja ventanas estrechas de tiempo y vehículos con capacidad pequeña, las de tipo 2 manejan ventanas grandes de tiempo y capacidad grande de vehículos. Se tienen tres clasificaciones C, R y RC, la clasificación C son las instancias cuya distribución territorial por cliente es arracimado, en las instancias de clase R los cliente están uniformemente distribuidos en un área territorial, las instancias de clase RC son la combinación de distribución territorial arracimada y distribuida. En total las

instancias que se utilizan comúnmente en el benchmark de Solomon para VRPTW son C1, R1, RC1, C2, R2, RC2 la Figura 2.2 muestra la caracterización de estas instancias.

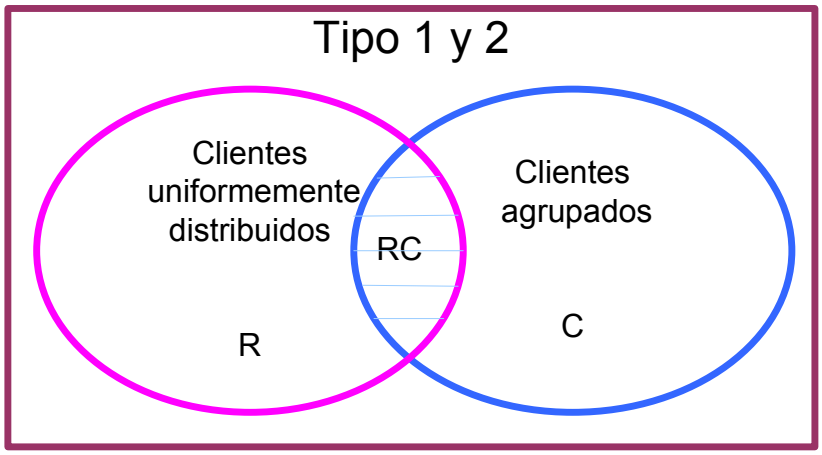

Figura2.2. Caracterización de instancias de RPTW

La representación numérica de las instancias para el problema del transporte con ventanas de tiempo contiene información del número de clientes, la ubicación geográfica de los clientes, la demanda por cliente, el tiempo de inicio de la ventana de tiempo, el tiempo final de la ventana de tiempo y el tiempo de servicio de la ventana de tiempo. La instancia más pequeña definida para este tipo de problema es la de 25 clientes para los dos tipos y las tres clasificaciones. La instancia más grande para la cual se ha encontrado algunas soluciones de algunas instancias es la de 100 clientes; existen instancias que actualmente se están trabajando de 200 clientes para las cuales se han encontrado algunas soluciones, pero aun no se han encontrado un óptimo. La Tabla 2.1 da un ejemplo de cómo se presentan los datos en literatura para ser usados en la experimentación de una instancia definida por Solomon para el problema de ruteo vehicular con

ventanas de tiempo tipo 1, clasificación C, para 25 clientes con una flotilla homogénea de 25 vehículos con capacidad de 200.

Tabla 2.1. Instancia definida por Solomon para el problema de ruteo vehicular con ventanas de tiempo

C101						
VEHICLE NUMBER 25			CAPACITY 200			
CUSTOMER No.	XCOORD	YCOORD	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	40	50	0	0	1236	0
1	45	68	10	912	967	90
2	45	70	30	825	870	90
3	42	66	10	65	146	90
4	42	68	10	727	782	90
5	42	65	10	15	67	90
6	40	69	20	621	702	90
7	40	66	20	170	225	90
8	38	68	20	255	324	90
9	38	70	10	534	605	90
10	35	66	10	357	410	90
11	35	69	10	448	505	90
12	25	85	20	652	721	90
13	22	75	30	30	92	90
14	22	85	10	567	620	90
15	20	80	40	384	429	90
16	20	85	40	475	528	90
17	18	75	20	99	148	90
18	15	75	20	179	254	90
19	15	80	10	278	345	90
20	30	50	10	10	73	90
21	30	52	20	914	965	90
22	28	52	20	812	883	90
23	28	55	10	732	777	90
24	25	50	10	65	144	90
25	25	52	40	169	224	90

Los datos que se presentan en la Tabla 2.1 dan información del tipo de instancia en este caso es la C101, que se lee instancia C tipo 1 numero 01. Que cuenta con 25 vehículos cada uno con capacidad de 200. CUSTOMER No. se refiere al número de clientes en este caso la instancia cuenta con 25 clientes XCOOD, Y COORD se refieren a las coordenadas donde está ubicado cada cliente, DEMAND es la demanda por cliente, READY TIME e

tiempo de inicio de la ventana de tiempo en que el cliente debe ser atendido, DUE DATE el tiempo final de la ventana de tiempo en que el cliente debe ser atendido y SERVICE TIME representa el tiempo de servicio dedicado a cada cliente para ser atendido.

Para ejemplificar el uso de los datos de las instancias se muestra una solución de una pequeña instancia de once clientes en la Figura 2.3.

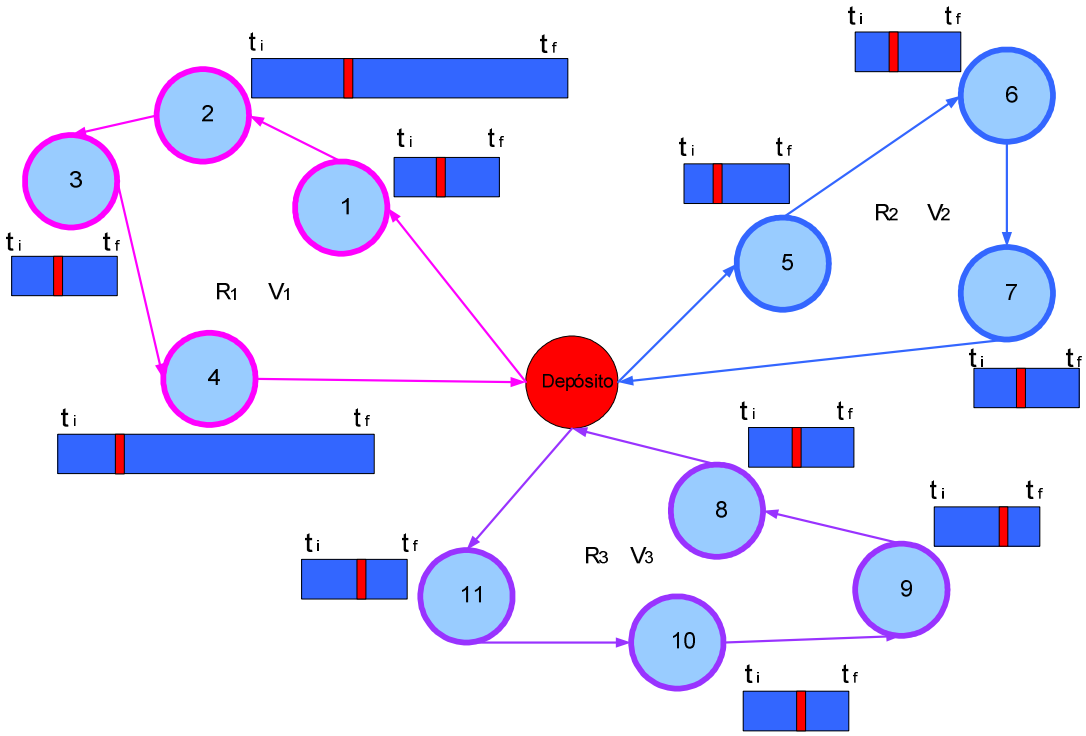

Figura2.3. Una solución con su representación de las ventanas de tiempo

La Figura 2.3 muestra una de las posibles soluciones que está compuesta de un conjunto de tres rutas S = {R_1, R_2, R_3}. La ruta R_1 atiende a los clientes 1, 2, 3 y 4; la ruta R_2 atiende a los clientes 5, 6 y 7; la ruta R_3 atiende a los clientes 8, 9, 10 y 11. El número de vehículos utilizados para la solución es de tres, uno por ruta, con capacidad de 200 cada uno. La instancia de once clientes se define en la Tabla 2.2.

Tabla 2.2. Instancia para 11 clientes

Número Cliente	Coordenada X	Coordenada Y	Demanda	Ready Time	Due Date	Tiempo Servicio
0	20	20	0	0	24	0
1	10	30	70	8	9	1
2	8	40	50	10	13	1
3	1	36	30	11	12	1
4	8	20	50	12	15	1
5	30	25	100	8	9	1
6	60	45	20	10	11	1
7	60	23	80	13	14	1
8	25	17	30	10	11	1
9	37	11	90	13	14	1
10	22	7	70	15	16	1
11	17	9	10	17	18	1

Para que se pueda construir una solución factible, se deben respetar las restricciones definidas del problema. La ruta R_1 atendida por el vehículo V1 cumple con las restricciones de demanda por cliente no mayor a 200 que es la capacidad del vehículo. Lo mismo para las rutas R_2 y R_3 atendidas por los vehículos V_2 y V_3 respectivamente. En lo que respecta a las ventanas de tiempo de cada cliente se debe atender primero el cliente con tiempo de inicio de atención menor a los demás clientes, en el caso de que exista más de un cliente con el mismo tiempo de inicio de atención será asignado a un vehículo diferente. Las ventanas de tiempo se representan en la Figura 2.3, en los rectángulos, las barras azules representan la ventana de tiempo. El área azul indica el tiempo disponible por el cliente (contemplando un tiempo de inicio t_i y un tiempo final t_f en un intervalo de atención). El área roja indicará el tiempo de inicio del servicio al cliente.

La calendarización de la solución de la instancia de once nodos, tomando en cuenta los tiempos de atención por cliente se muestran en el diagrama de Gantt de la Figura 2.4.

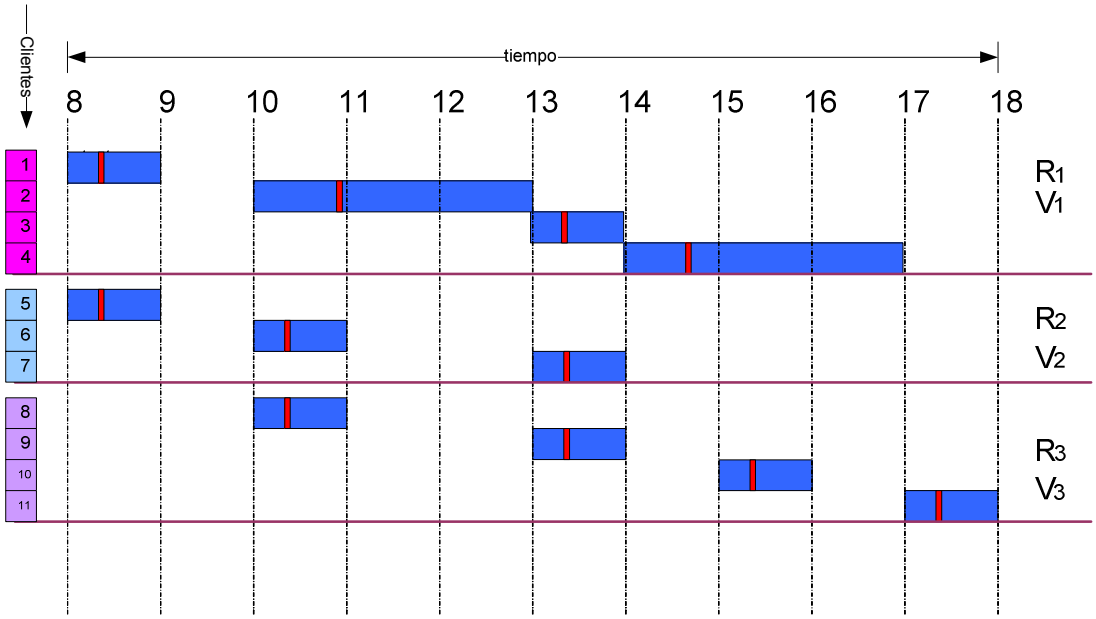

Figura 2.4. Calendarización de instancia de once clientes representando los tiempos de ventana por cliente

La Figura 2.4 muestra los tiempos sin traslapes en los que cada vehículo debe atender a los clientes de la ruta implicada. Para la ruta R1 el tiempo total de recorrido de la ruta será la suma del tiempo representado por los rectángulos rosas. Para la ruta R2 la suma del tiempo representado por los rectángulos azul claro y para la ruta R3 la suma del tiempo representado por los rectángulos morados.

La restricción de ventanas de tiempo por cliente hace a esta variante del problema VRP difícil de resolver. Para poder clasificar los problemas en base a la complejidad que implica el encontrar una solución, existe la teoría de la complejidad que estudia si un problema es tratable, es decir si es posible encontrar una solución [Hopcroft y Ullman, 1993] [Cortéz, 2004] [Garey y Johnson, 1979]. Según la teoría de la complejidad el problema del transporte está clasificado dentro de los problemas NP-completos [Lenstra y Rinnoy, 1981] [Savelsbergh, 1985]. En la siguiente sección se describe la complejidad del problema de ruteo vehicular con ventanas de tiempo.

2.3.1. Complejidad del problema del transporte con ventanas de tiempo

La teoría de la complejidad clasifica al VRPTW dentro de la denominada clase de problemas NP-Completo [Savelsbergh, 1985] [Balakrishnan, 1993] [Larsen, 2001]. Esto significa que el esfuerzo de computación que se ha de realizar para encontrar una solución óptima crece de forma exponencial conforme aumenta el tamaño del problema. El espacio de soluciones del problema de ruteo vehicular con ventanas de tiempo es acotado como (n-1)!. El comportamiento del espacio de soluciones se representa en la Figura 2.5. Donde n representa el número de clientes.

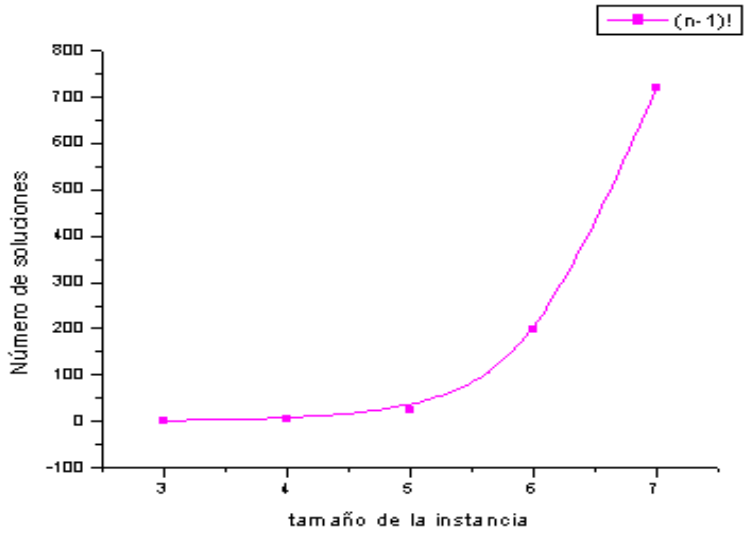

Figura 2.5. Comportamiento del espacio de soluciones (n-1)!

Debido a la complejidad del problema de ruteo vehicular con ventanas de tiempo es necesario emplear métodos aproximados de manera que se puedan encontrar soluciones suficientemente buenas en un tiempo de computación razonable.

Las soluciones propuestas para este tipo de problemas deben ser algoritmos con complejidad logarítmica o polinomial, para que el tiempo en encontrar una solución sea razonable ya que algoritmos con complejidades mayores implican mayor tiempo de computación.

Para poder proponer una solución a un problema se debe empezar por conocer como se define éste y las variables implicadas en su solución. Para analizar problemas más a detalle y conocer las principales variable que lo afectan existen algunas metodologías de análisis como la metodología Chekland [Díaz-Parra et al., 2006] que de inicio fue utilizada para el análisis del problema del transporte con ventanas de tiempo para conocer principalmente que variables lo afectan más. En la siguiente sección se define el modelo que representa el problema de ruteo vehicular con ventanas de tiempo y las variables implicadas.

2.3.2. Modelo matemático del problema del transporte con ventanas de tiempo

Como todo problema de optimización, el modelo VRPTW [Toth y Vigo, 2001] se representa por la función objetivo en la ecuación (1) y el conjunto de restricciones en las ecuaciones (2) a (11).

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{1}$$

$$\sum_{k \in K} \sum_{j \in \Delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N \tag{2}$$

$$\sum_{j\in\Delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K \tag{3}$$

$$\sum_{i\in\Delta^-(j)} x_{ijk} - \sum_{i\in\Delta^+(j)} x_{ijk} = 0 \qquad \forall k \in K, j \in N \tag{4}$$

$$\sum_{i\in\Delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K \tag{5}$$

$$w_{ik} + s_i + t_{ij} - w_{jk} \le (1 - x_{ijk})M_{ij} \qquad \forall k \in K, (i,j) \in A \tag{6}$$

$$a_i \sum_{j\in\Delta^+(i)} x_{ijk} \le w_{ik} \le b_i \sum_{j\in\Delta^+(i)} x_{ijk} \qquad \forall k \in K, i \in N \tag{7}$$

$$E \le w_{ik} \le L \qquad \forall k \in K, i \in (0, n+1) \tag{8}$$

$$\sum_{i\in N} d_i \sum_{j\in\Delta^+(i)} x_{ijk} \le C \qquad \forall k \in K \tag{9}$$

$$x_{ijk} \ge 0 \qquad \forall k \in K, (i,j) \in A \tag{10}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A \tag{11}$$

La nomenclatura del modelo es la siguiente:

c	Representa el costo del servicio
x	Representa el si se ejecuta la operación
k	Representa un vehículo
K	Representa la flota de vehículos
A	Representa conjunto formado por los pares ordenados *i,j*.
i	Representa nodo origen
j	Representa nodo destino
$j \in \Delta^+(i)$	Significa que sea un destino *j* unido a un origen *i* con dirección de *i* a *j*.

$i \in \Delta^{-}(j)$	Significa que sea un origen *i* unido directamente a un destino *j* con dirección de *j* a *i*.
$n+1$	Representa el depósito
w	Representa el tiempo de inicio del servicio en el nodo
a	Representa el tiempo de inicio de la ventana en un nodo
b	Representa el tiempo límite de la ventana en un nodo
E	Representa el tiempo de inicio de la ventana en el depósito
L	Representa el tiempo límite de la ventana en el depósito
d	Representa la demanda por nodo
C	Representa la capacidad del vehículo
N	Representa el conjunto de nodos

En la ecuación (1). La función objetivo implica el costo total de la ruta a seguir por cada una de las unidades vehiculares *k.* El costo total se puede interpretar como el recorrido total en unidades de distancia, tiempo o costo. En este modelo lo que se busca es minimizar el costo total del recorrido y el número de vehículos implicados en el recorrido total. Todo recorrido *x* inicia y termina en el depósito. El depósito está representado por el nodo *i* cuando *i=0* y el vehículo *k* inicia el recorrido ó por *i=n+1* cuando el vehículo *k* regresa al depósito al terminar el recorrido. El conjunto de restricciones que deben cumplirse se presentan en las ecuaciones (2) a (11).

En la ecuación (2). El conjunto de restricciones representadas indican que debe ser asignado un nodo a una ruta. El recorrido *x* del nodo *i* a *j* que forma la ruta debe ser igual a 1 para garantizar un vehículo por ruta.

En la ecuación (3). El conjunto de restricciones garantiza que el número de clientes *j* que son directamente alcanzables desde el depósito *0* utilizando el vehículo *k*, sea igual a uno. Partiendo del depósito sólo un nodo destino puede ser atendido.

En la ecuación (4). El conjunto de restricciones garantiza que el recorrido *x* realizado por el vehículo *k* de un nodo origen *i* directamente alcanzable, a un

nodo destino *j* debe ser igual al recorrido *x* de un nodo origen *j* directamente alcanzable, a un nodo destino *i*.

En la ecuación (5) El conjunto de restricciones garantiza que para cada vehículo k solo debe existir un nodo que conecte directamente con el depósito n+1

En la ecuación (6). El conjunto de restricciones garantiza la atención del nodo i antes del nodo j. Considerando que la suma de el tiempo de inicio de servicio al nodo i w_{ik} , el tiempo de servicio asignado al nodo i, el tiempo de recorrido del nodo i al nodo j y el tiempo de inicio de servicio del nodo j w_{jk} , debe ser menor o igual a ($1-x_{ij}$). Donde M es una constante con valores muy grandes.

En la ecuación (7). El conjunto de restricciones indican que el tiempo de inicio de servicio w_{ik} a cada nodo no debe exceder los límites de la ventana de tiempo $[a_i, b_i]$.

En la ecuación (8). El conjunto de restricciones indican que el tiempo de inicio de servicio w_{0k} en el depósito no debe exceder los límites de la ventana de tiempo [*E, L*].

En la ecuación (9). El conjunto de restricciones representadas en la desigualdad es la suma de las demandas *d* de cada nodo en el recorrido *x*, que no debe exceder la capacidad total *C* del vehículo. Donde *N* indica el número de clientes a atender, *K* indica la flotilla de vehículos con capacidad *C.*

En la ecuación (10). El conjunto de restricciones garantizan la no negatividad de las variables *x*.

En la ecuación (11). El conjunto de restricciones garantiza que el modelo lineal se defina como un modelo lineal entero binario.

El conjunto de restricciones mencionadas y la función objetivo deben tomarse en cuenta para construir una solución al problema de ruteo vehicular. En la siguiente sección se citan a los principales autores que han trabajado en la construcción de soluciones para este tipo de problema utilizando diferentes técnicas heurísticas.

2.3.3. Trabajos relacionados

Existen trabajos realizados para buscar solución al problema de ruteo vehicular con ventanas de tiempo utilizando heurísticas como algoritmos genéticos [Chin et al., 1999] [Louis et al., 1999] [Maeda et al., 1999] [Tan et al., 2001], colonias múltiples de hormigas [Gambardella et al., 1999 b], hibridación de colonia de hormigas y genéticos [Hermosilla y Barán, 2005], búsqueda tabú [Potvin et al., 1996], recocido simulado [Arbelaitz et al., 2001].

Otra de las alternativas de solución al problema de rutero vehicular trabajada por varios autores son las heurísticas paralelizadas, por ejemplo Gupta propone algoritmos paralelos con un número polinomial de procesadores [Gupta. y Krishnamurti, 1997], otro caso es el de Arbelaitz que propone un algoritmo de recocido simulado paralelo [Arbelaitz et al., 2001], [Czech y Czarnas, 2002]. Los trabajos realizados por Gupta, Arbelaitz y Czech analizan el problema en cuestión en un contexto mono-objetivo, aunque en algunos casos se proponen priorizaciones y/o combinaciones de objetivos, sin llegar a calcular el conjunto completo de soluciones. La importancia de los algoritmos heurísticos radica en la ejecución en tiempo

polinomial y la aproximación de soluciones. Bräysy describe la importancia en tiempo de las heurísticas y meta heurísticas comúnmente estudiadas en literatura [Bräysy y Gendreau, 2001]. Dentro de todo el conjunto de heurísticas y meta heurísticas sólo algunas generan buenos resultados para el problema del ruteo vehicular, en la siguiente sección se muestran las heurísticas que mejor resuelven el problema de ruteo de vehículos con ventanas de tiempo.

2.3.3.1. Algoritmos que mejor resuelven el problema

Entre las técnicas más comunes de solución del VRPTW se encuentran los métodos heurísticos, porque los métodos exactos de solución no garantizan encontrar la solución óptima en un tiempo razonable de computación cuando el número de clientes es grande [Arbelaitz et al., 2001]. Por lo que en literatura se han propuesto varios métodos heurísticos para la solución del problema del transporte con ventanas de tiempo [Tan et al., 2000] [Martí y Cunquero, 2003]. Los métodos comúnmente usados son los algoritmos de hormigas, de recocido simulado, de búsqueda tabú, de genéticos. Algunos autores que han trabajado soluciones al problema de ruteo vehicular con ventanas de tiempo son: [Olivera, 2004], quien presenta una solución mediante búsqueda Tabú, [Gendreau et al., 1998] proponen una heurística de inserción; [Vacic, 2002] [Bräysy, 2001] [Zhu , 2000] y [Louis et al., 1999] lo resuelven con algoritmos genéticos y [Baran y Schaerer, 2003] y [Gambardella et al., 1999] presentan una propuesta a través de algoritmos de colonia de Hormigas. A continuación se muestra una representación gráfica de algunos de los algoritmos heurísticos comúnmente utilizados para resolver el problema del transporte.

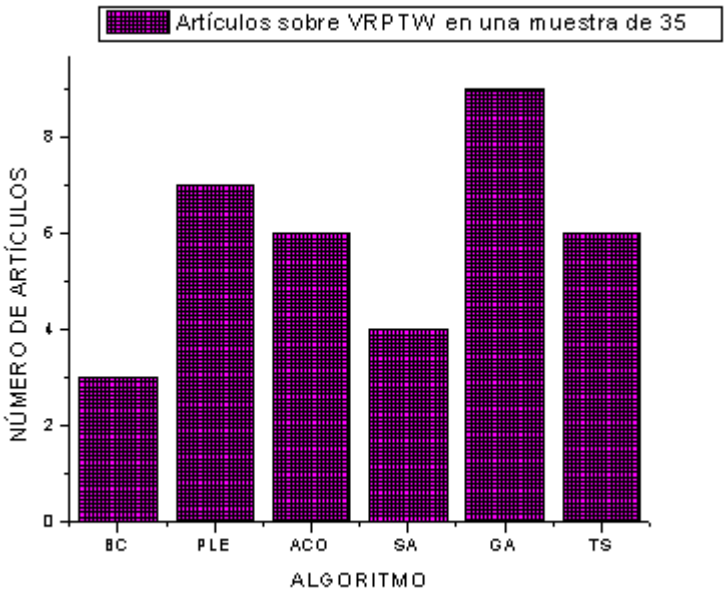

Figura 2.6. Gráfica de diferentes heurísticas tomadas de una muestra de 35 artículos

En la Figura 2.6 se presenta un número de artículos con la heurística utilizada para resolver el problema de ruteo de vehículos con ventanas de tiempo. Donde BC (por sus siglas en inglés, Branch and Cut) se refiere al algoritmo ramificación y acotamiento, ILP (por sus siglas en inglés, Integer Lineal Programming) al algoritmo de programación lineal entera, ACO (por sus siglas en inglés, Ant Colony Optimization) al algoritmo de hormigas, SA (por sus siglas en inglés, Simulated Annealing) al algoritmo de recocido simulado, GA (por sus siglas en inglés, Genetic Algorithm) al algoritmo genético y TS (por sus siglas en inglés, Tabú Search) al algoritmo de búsqueda tabú. Las heurísticas comúnmente utilizadas son: Búsqueda tabú [Bräysy y Gendreau, 2001] [Pacheco y Delgado, 2001] [Bräysy y Gendreau, 2002] [Pacheco y Marti, 2006] [Pacheco et al., 2000] [Delgado y Pacheco, 2001], recocido simulado [Czech y Czarnas, 2002] [Olivera, 2004] [Chong, 2004] [Brando et al., 2006] [Thangiah, 1999] [Olatz y Gallego, 200], Colonia de hormigas [Hermosilla y Barán, 2005] [Balseiro, 2006] [Baran y Schaerer,

2003] [Balseiro, 2007] [Gambardella et al., 1999 b], mecanismos de búsqueda local [Pacheco y Delgado, 2000] [Bräysy y Gendreau, 2001 b] [Alegre et al., 2002] y algoritmos genéticos [González y Aristizábal, 2006] [González y Aristizabal, 2007] [Alba et al., 2003] [Bastos et al., 2005] [Berger et al., 2004] [Ombuki et al., 2004] [Thangiah, 1995]. En una muestra tomada de 35 artículos, publicados en los últimos siete años, nueve de ellos utilizan algoritmos genéticos como solución al problema de ruteo vehicular con ventanas de tiempo. Esto representa un 25% de utilización de algoritmos genéticos en la construcción de posibles soluciones. El análisis de esta muestra de artículos deja clara la tendencia de obtención de soluciones mediante algoritmos evolutivos. Esta tendencia por los algoritmos genéticos es por su característica de exploración y explotación de espacio de soluciones grandes y obtención de óptimos globales. Se denomina exploración del espacio de soluciones al proceso de visitar nuevas regiones del espacio de búsqueda para tratar de encontrar soluciones. La exploración involucra grandes saltos en el espacio de búsqueda y es un mecanismo útil para evitar que un algoritmo quede atrapado en óptimos locales. Una explotación del espacio de soluciones es el proceso de utilizar la información obtenida de puntos del espacio de búsqueda previamente visitados para determinar los puntos que conviene visitar. La explotación involucra movimientos finos y es un mecanismo provechoso para que un algoritmo encuentre óptimos locales. En un algoritmo genético en el operador de cruzamiento es donde se lleva a cabo la exploración y en el operador de mutación la explotación del espacio de búsqueda.

En este trabajo de investigación se propone una alternativa de solución mediante un algoritmo evolutivo en combinación con búsquedas locales en vecindades que sea eficaz y eficiente y que compita con los mejores existentes.

Un algoritmo evolutivo engloba una serie de técnicas inspiradas biológicamente en los principios de la teoría Neo-Darwiniana de la evolución natural. Dentro del este tipo de algoritmos encontramos a los algoritmos genéticos originalmente llamados “planes reproductivos genéticos” propuestos por Holland [Holland, 1962] [Holland, 1962 b] un algoritmo genético definido por Koza es “un algoritmo matemático que transforma un conjunto de individuos o población (colección de objetos matemáticos representando un individuo), cada uno de los cuales tiene asociado un valor de adaptación, en una nueva población (la siguiente generación) utilizando una serie de operadores basados en los principios darwinianos de supervivencia del más adaptado” [Koza, 1992] [Coello, 2008]. Se compone básicamente de operadores genéticos de selección, cruzamiento y mutación. El operador de selección escoge a individuos de la población, el operador de cruzamiento enfatiza la cruza sexual entre individuos y el operador de mutación realiza cambios entre genes de cada individuo.

En literatura existen diferentes operadores genéticos, propuestos por varios autores, por ejemplo: para el operador de selección [Holland, 1975] encontramos la técnica de la ruleta [Dejong, 1975], que consiste de dar a cada individuo una probabilidad de ser seleccionado de acuerdo a su función de costo, cuanto mejor sea la función de costo mayor será la probabilidad de que sea seleccionado, se generan tantos números aleatorios como individuos queramos seleccionar. Cada número aleatorio se compara con las probabilidades acumuladas y se escoge al individuo con probabilidad inmediatamente menor al número aleatorio. Otra técnica de selección es la de torneo [Wetzel, 1983] que consiste en tomar dos individuos aleatoriamente de la población y se genera un número aleatorio r entre cero y uno. Si $r<k$, donde k es un parámetro, se selecciona el mejor de los individuos en caso contrario se selecciona el peor, los dos individuos se devuelven a la población inicial para que puedan ser seleccionados

nuevamente. Existe también la técnica de selección “the best” que consiste en seleccionar el mejor cromosoma (según lo determinado por la aptitud) si existen dos o más cromosomas con la misma mejor aptitud uno de ellos se elige aleatoriamente [Reid et al., 1991].

Para el operador de cruzamiento existe el cruce de un punto que consiste en elegir un punto de corte aleatoriamente y se intercambian las secciones que se encuentran a la derecha de dicho punto [Holland, 1975]. Otra técnica de cruzamiento es el cruce basado en múltiples puntos que consiste en realizar varios cortes aleatoriamente. Una derivación del operador de cruce múltiple es el operador de cruce de dos puntos que consiste en realizar dos cortes aleatoriamente de un segmento [DeJong, 1975]. Existe también el operador de cruce uniforme que consiste en que cada gen en la descendencia se crea copiando el correspondiente gen de uno de los dos padres, escogido de acuerdo a un número binario generado aleatoriamente llamado máscara de cruce. Cuando existe un 1 en la máscara de cruce el gen es copiado del primer padre mientras que cuando exista un cero en la máscara el gen se copia del segundo padre [Syswerda, 1991].

Para el operador de mutación existen varias técnicas: la mutación por inserción, que consiste de seleccionar un valor en forma aleatoria e insertarlo en una posición arbitraria. La mutación por desplazamiento, que consiste en generar varios valores aleatoriamente y realizar varios movimientos en posiciones arbitrarias [Holland, 1975]. La mutación heurística, que consiste en seleccionar un conjunto de genes al azar, generar todas las posibles permutaciones de los genes seleccionados y evaluarlos para escoger el mejor, de la mejor permutación se insertan los genes en las posiciones correspondientes de donde fueron tomados los genes inicialmente [Gen y Cheng, 2000]. La mutación flip-bit, consiste en la inversión del valor de un digito binario en un cromosoma. La inversión del valor del digito tiene lugar en aquellas posiciones en donde un cierto número

aleatorio comprendido entre cero y uno, obtenido para dicha posición, es inferior o igual a la probabilidad de mutación [Lahoz-Beltra, 2005]. El operador tradicional de mutación introduce diversidad en el mecanismo evolutivo, solamente por realizar la inversión del valor de un bit (flip bit) [Nesmachnow, 2004].

En el capítulo tres se explica la propuesta de solución al problema de ruteo vehicular planteada por el autor de este libro.

Capítulo 3. Algoritmo evolutivo secuencial para el problema de ruteo de vehículos con ventanas de tiempo

En este capítulo se describe la metodología utilizada para la selección del tipo de heurística, la descripción a detalle y las mejoras realizadas a la heurística propuesta.

Para seleccionar el tipo de heurística se realizó una búsqueda en publicaciones que abordan el problema de ruteo de vehículos con ventanas de tiempo (mencionado en el capítulo 2 de este trabajo) del cual se obtuvo información sobre las heurísticas comúnmente utilizadas y las que mejor resuelven el problema de ruteo de vehículos con ventanas de tiempo. Como primer acercamiento a una solución, se seleccionó una heurística evolutiva y como mejora a la eficacia del algoritmo evolutivo se hibridó con una búsqueda local en vecindades. En la siguiente sección se muestra la metodología de construcción del algoritmo híbrido secuencial.

3.1. Metodología de construcción del algoritmo híbrido secuencial

Para proponer una solución al problema de ruteo de vehículos con ventanas de tiempo se realizó un análisis de artículos publicados que abordan el problema y sus propuestas de solución con algún tipo de heurística. Resultando la heurística de los algoritmos genéticos la más frecuente utilizada para este tipo de problema. Una de las características de los algoritmos genéticos es que exploran y explotan el espacio de búsqueda mediante los operadores genéticos de cruzamiento y mutación respectivamente.

Partiendo de este análisis se propone la hibridación de un algoritmo genético con búsqueda local, para conseguir una mayor explotación del espacio de búsqueda. La construcción del algoritmo propuesto se deriva de varias versiones realizadas para finalmente elegir la mejor. En la primera versión GA-PCP (por sus siglas en ingles, Genetic Algorithm with Precedent Constraint Posting) se utilizó la técnica de fijar la relación de precedencia PCP (por sus siglas en inglés, Precedent Constraint Posting) [Cruz-Chávez et al., 2008 b] [Cruz-Chávez et al., 2007]. En la segunda versión GA-VRPTW (por sus siglas en ingles, Genetic Algorithm for Vehicle Routing Probem with Time Windows) se propuso para la búsqueda local un operador de mutación inteligente [Díaz-Parra y Cruz-Chávez, 2008]. En la tercera versión GA-VRPTW-SN (por sus siglas en ingles, Genetic Algorithm for Vehicle Routing Probem with Time Windows Secuencial Neighborhood) se propuso trabajar la mutación con un mecanismo de vecindad [Cruz-Chávez et al., 2008]. En la cuarta versión GA-VRPTW-PN (por sus siglas en ingles, Genetic Algorithm for Vehicle Routing Probem with Time Windows Parallel Neighborhood) para hacer eficiente el algoritmo se propuso paralelizar la versión 3 aplicando una metodología para convertir algoritmos secuenciales a paralelos [Díaz-Parra y Cruz-Chávez, 2008 b]. Resultando ésta última versión la de mejor comportamiento eficaz y eficiente para el problema del transporte con ventanas de tiempo.

En la metodología de construcción del algoritmo propuesto básicamente se utiliza la heurística GA y se adapta para el problema de ruteo de vehículos con ventanas de tiempo, se le da como entrada instancias tipo 1 para 25 y 100 nodos de las propuestas por Solomon (definidas en el capítulo 2 de este documento). Posteriormente se toma un algoritmo de búsqueda local (LS) y se combina con el algoritmo GA. Para cada una de las versiones la búsqueda local utilizada es diferente excepto en las versiones 3 y 4. En la Figura 3.1 se ilustra la metodología de construcción para el algoritmo

propuesto. Donde PCP, LS y N representan las técnicas de búsqueda local utilizadas en las diferentes versiones y GA se refiere al algoritmo genético. Cabe mencionar que en este documento solo se explicarán las versiones 2, 3 y 4 del algoritmo. En la siguiente sección se explicará a detalle los operadores genéticos utilizados en la versión 2 y la técnica de búsqueda local utilizada en la versión 3. Para mayor detalle de las versiones 1 y 2 consultar la referencia [Cruz-Chávez et al., 2008 b] y [Díaz-Parra y Cruz-Chávez, 2008].

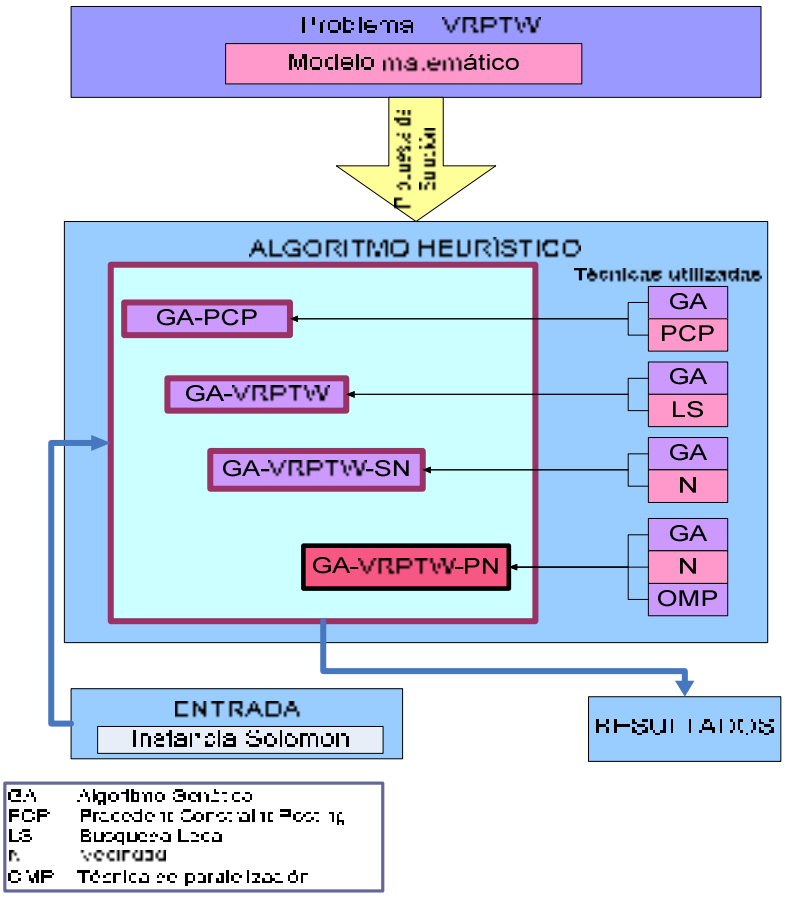

Figura 3.1. Metodología de construcción del algoritmo propuesto

De todas las versiones del algoritmo construidas, la versión 4 (GA-VRPTW-PN) representa la solución propuesta por el autor. Que básicamente se construyó partiendo de la versión 3 (GA-VRPTW-SN) que es la versión

secuencial del algoritmo. En la siguiente sección se explica la manera como se construyó el algoritmo evolutivo secuencial, así como cada uno de los elementos que lo componen.

3.2. Algoritmo evolutivo secuencial

La versión 3 (GA-VRPTW-SN) es la versión secuencial del algoritmo propuesto como solución al problema del transporte con ventanas de tiempo. Para construir la versión 3 se tomaron conceptos importantes de la versión 2 que se explica a continuación. El algoritmo genético de la versión 2 (GA-VRPTW) se construye utilizando operadores de selección, cruzamiento y mutación. El algoritmo evolutivo GA-VRPTW utiliza como operador de selección la estrategia de el mejor (*the best*), el operador de cruzamiento utilizado es una adecuación del operador de cruzamiento basado en dos puntos propuesto por DeJon [DeJong, 1975] dando como resultado un cruzamiento llamado por el autor "*cruzamiento-k"* y un operador de mutación con adecuación a las restricciones del problema de ruteo de vehículos con ventanas de tiempo llamado por el autor "*mutación-S"*. La población inicial se genera utilizando un mecanismo de agrupación comúnmente utilizado en minería de datos llamado k-means [Maimon y Rokach 2005] [Villagra et al., 2007]. Esta combinación de operadores se obtuvo de la realización de varias pruebas utilizando diferentes operadores genéticos utilizando instancias de 25 nodos de las propuestas por Solomon. Las diferentes combinaciones realizadas se muestran en la Tabla 3.1

Tabla 3.1. Diferentes combinaciones de operadores genéticos

	Población	Selección	Cruzamiento	Mutación
1	Aleatoria	Ruleta (literatura)	Un punto (literatura)	Flip-bit(literatura)
2	Factibles	Torneo (literatura)	Un punto (literatura)	Flip-bit(literatura)
3	Factibles	The best (literatura)	Un punto (literatura)	Flip-bit(literatura)
4	Factibles con PCP	The best (literatura)	Cruza-k (propuesto)	Flip-bit(literatura)
5	Factibles con PCP	Torneo (literatura)	Cruza-k (propuesto)	Flip-bit(literatura)
6	Agrupación k-means Factibles	The best (literatura)	Cruza-k (propuesto)	Muta-S (propuesto)

Donde los números de 1 al 6 representan el número de la combinación de operadores genéticos (población, selección, cruzamiento y mutación). Nótese que en las combinaciones del 1 al 5 el operador de mutación es el mismo. La combinación de operadores genéticos tomados tal cual se definen en literatura, representados en las combinaciones del 1 al 3, no generó resultados esperados al problema de ruteo de vehículos con ventanas de tiempo. Por lo que se vio la necesidad de generar las combinaciones 4 a 6, siendo la combinación número 6 la que mejor resultó para el problema de ruteo vehicular con ventanas de tiempo. El algoritmo genético con esta combinación se presenta en la Figura 3.2. Donde los operadores modificados fueron el cruzamiento y la mutación. La selección se utiliza según la definen en literatura y para la población inicial se incorporó un mecanismo de agrupación llamado *k-means*.

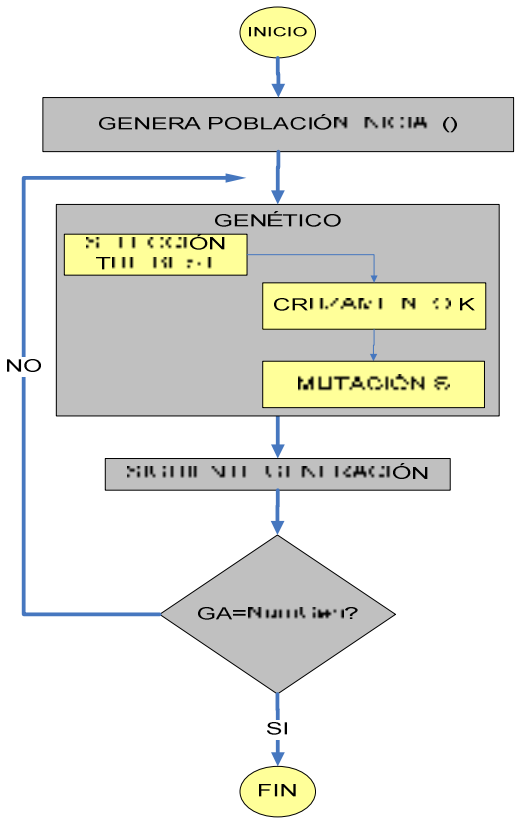

Figura 3.2. Algoritmo evolutivo secuencial GA-VRPTW

El algoritmo genético presentado en la Figura 3.2 utiliza como criterio de paro el número de generaciones. En la siguiente sección se describen cada uno de los operadores utilizados en la combinación 6 (agrupación *k-means*, selección *the best*, *cruzamiento-K y mutacion-S*).

3.2.1. Descripción de los operadores genéticos utilizados

De las diferentes combinaciones de operadores genéticos realizadas se observa que las combinaciones entre operadores genéticos comúnmente utilizados en literatura (1 al 3 de la Tabla 3.1) no generaron buenos resultados para el problema de ruteo de vehículos con ventanas de tiempo.

Por lo que se probó con otras combinaciones entre operadores tomados de literatura y los propuestos por el autor. La combinación de operadores genéticos que mejor funciona en versión secuencial (para esta propuesta) es la número 6 mostrada en la Tabla 3.1 con los operadores de *cruzamiento-k* y *mutación-S*, propuestos por el autor. A continuación se describen a detalle cada uno de los operadores genéticos utilizados en la versión GA-VRPTW del algoritmo secuencial.

La técnica de agrupación *k-means* [McQueen, 1967] utiliza a los centroides de cada grupo como sus puntos representantes. Partiendo de una selección inicial de *k* centroides (que pueden ser *k* elementos de la colección seleccionados al azar, o los que se obtengan mediante la aplicación de alguna técnica de inicialización), cada uno de los elementos de la colección se asigna al grupo con el centroide más cercano.

El operador de selección *the best* consiste en ordenar de menor a mayor la lista de individuos de la población y dividirla en dos, se elije la primera mitad y se va seleccionando al mejor individuo (según lo determinado por la aptitud) si existen dos o más individuos con la misma mejor aptitud uno de ellos se elige aleatoriamente, se repite el proceso hasta completar la población deseada [Reid et al., 1991].

El operador de cruzamiento, llamado *cruzamiento-k* consiste en encontrar dos puntos aleatoriamente en el individuo (rand1 y rand2) y buscar en el individuo los genes correspondientes y hacer el cruzamiento, con esto se garantiza cumplir con una de las restricciones del problema VRPTW de no pasar dos veces por el mismo nodo en la Figura 3.3 se representa el mecanismo de funcionamiento del operador de *cruzamiento-k*.

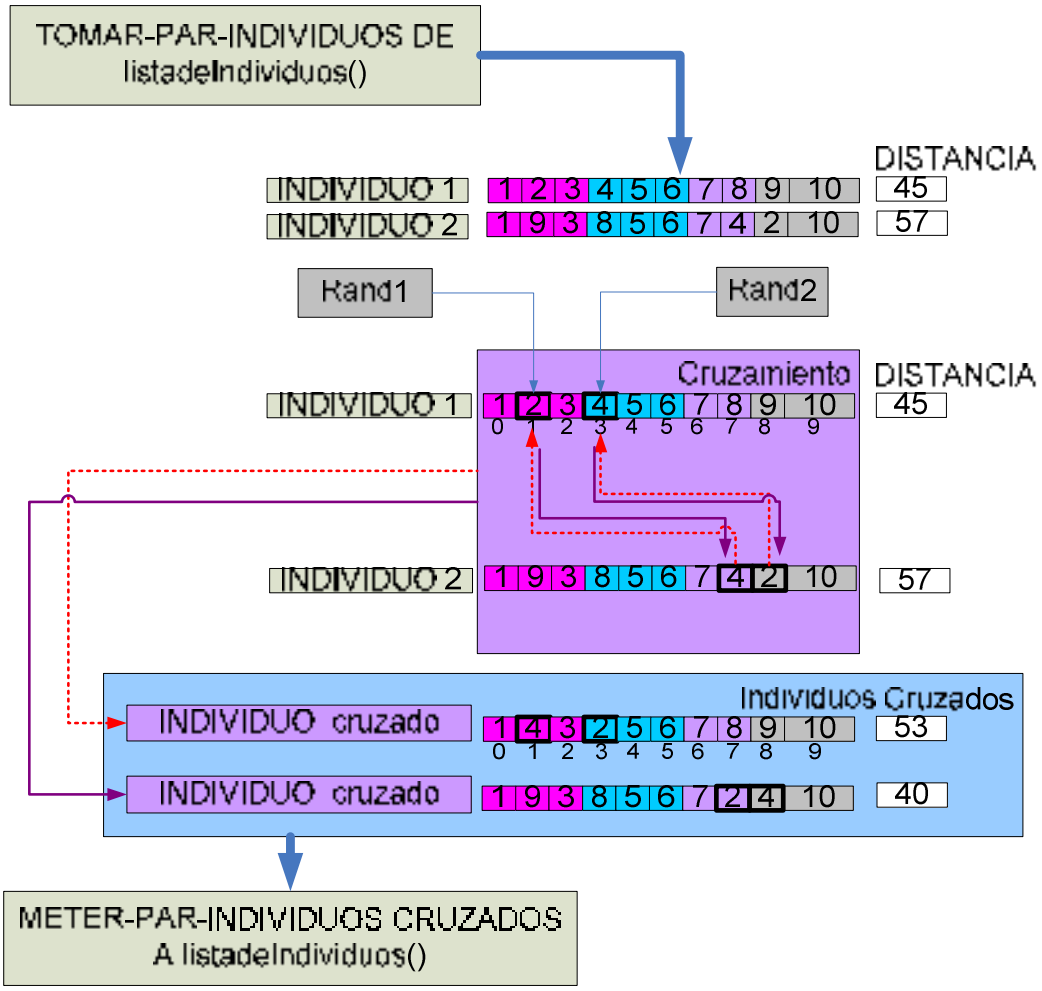

Figura 3.3. Mecanismo de funcionamiento del operador *cruzamiento-K*

El operador de *cruzamiento-K* genera dos individuos a partir de dos individuos. Esto es, se toman dos individuos aleatoriamente de la población, se generan dos números aleatorios y se realiza la búsqueda de los nodos correspondientes a los números aleatorios en el individuo 1. Una vez identificados los nodos a cruzar estos se buscan en el individuo 2. Ya que se han identificado las posiciones de los nodos correspondientes en los dos individuos se realiza el cruce de nodos en el individuo 1 y se genera el primer individuo cruzado, para generar el segundo individuo cruzado se realiza el cruzamiento en el individuo 2. El cruzamiento se realiza siempre y cuando los nodos a cruzar sean diferentes.

El operador de mutación, llamado mutación-S consiste en detectar cual de todos los genes que conforman el individuo es el que involucra mayor distancia. En base al gen de mayor distancia realiza una búsqueda en la matriz de distancias euclídeas con cada uno de los genes involucrados en el individuo, el que genere menor distancia es el gen candidato a mutar, seguidamente de identificar el gen candidato, verifica las estricciones de ventana de tiempo y capacidad de vehículo, si no se violan las restricciones entonces procede a realizar la mutación. El mecanismo de funcionamiento se muestra en la Figura 3.4. Lo que caracteriza al mecanismo de *mutacion-S* es que los movimientos a realizar no son aleatorios como los que comúnmente se utilizan en literatura, antes de realizar cualquier movimiento debe asegurarse de que el movimiento que se realice sea el que mejor convenga y que no viole las restricciones del problema. El *mutador-S* toma un individuo cruzado de la población e identifica el nodo con mayor distancia. Comienza la búsqueda de menores distancias entorno al nodo identificado como el de mayor distancia con todos los demás nodos que forman el individuo de tal forma que se identifique al nodo que no viole restricciones y que al menos tenga una distancia menor al nodo en cuestión. En el caso que no se encuentre un nodo que cumpla con las condiciones no se realiza la mutación y se procede a tomar otro individuo de la población.

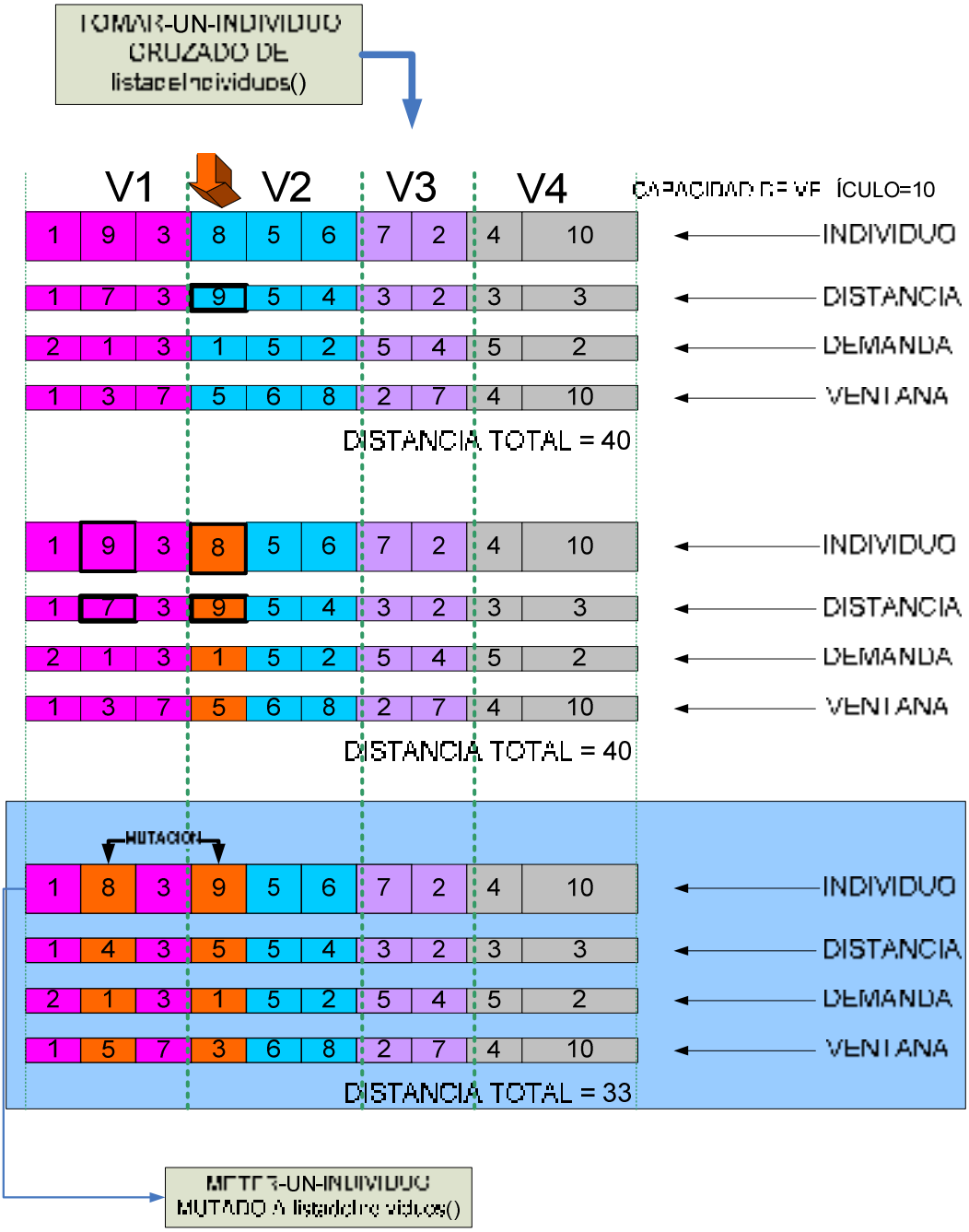

Figura 3.4. Mecanismo de funcionamiento del operador *mutación-S*

De la combinación de operadores de la versión 2 sólo el operador de *cruzamiento-K* es el que se implementó en la versión 3. En la siguiente

sección se describe la búsqueda local aplicada a la versión 3 (GA-VRPTW-SN).

3.2.2. Aplicación de búsqueda en vecindad

En la versión 3 del algoritmo GA-VRPTW-SN se propone la hibridación del algoritmo genético con un mecanismo de búsqueda en vecindades. La vecindad de una solución se define como el conjunto de todas aquellas soluciones que pueden ser alcanzables a partir de una solución *s'* por medio de un movimiento σ [Martínez-Morales, 2006], un movimiento puede ser un intercambio entre elementos que conforman la solución *s* [Cruz-Chávez et al., 2008]. En la ecuación 3.1 se muestra la definición de vecindad de una solución.

$$N(s) = \{s' \in S : s \xrightarrow{\sigma} s'\} \tag{3.1}$$

Donde *N(s)* representa la vecindad con respecto a *s, s* representa una solución tomada del espacio total de soluciones *S* y *s'* representa el vecino de *s* generado a partir de σ movimientos. Un movimiento puede ser una inserción, eliminación o intercambio de componentes en una solución. Para este caso en particular un movimiento estará definido como un intercambio de dos genes en un individuo.

El mecanismo de vecindad aplicado al algoritmo genético se describe en la Figura 3.5 tomando como muestra de vecinos un tamaño inicial de 10 y el criterio de paro de búsqueda local se tomó de inicio dos veces el tamaño de la vecindad. En la ecuación 3.2 se define el tamaño de la vecindad como $T_{N(s)}$. Donde $t_{(s)}$ representa el tamaño del individuo y $n_{(s)}$ representa el número de genes a mutar.

$$T_{N(s)} = \frac{t_{(s)}!}{n_{(s)}!(t_{(s)} - n_{(s)})!} \tag{3.2}$$

El tipo de vecindad aplicada a la versión 3 es la de tipo Or-modificada presentada en la Figura 3.5. Donde se inicia con una solución *x* a partir de la cual se genera el vecindario *N(x)*, se genera el conjunto se soluciones vecinas, con base al tamaño de la muestra λ. Se realiza la evaluación de la función objetivo si *s'* resulta menor que la solución inicial *x*. entonces a partir de *s'* se genera el próximo vecindario y así sucesivamente hasta cumplir con el criterio de paro de dos veces el tamaño de la vecindad. Este mecanismo de vecindad se probó experimentalmente con variaciones en el tamaño de la muestra de vecinos, para mas referencia del mecanismo de vecindad tipo Or-modificada véase [Barreto-Sedeño, 2008]

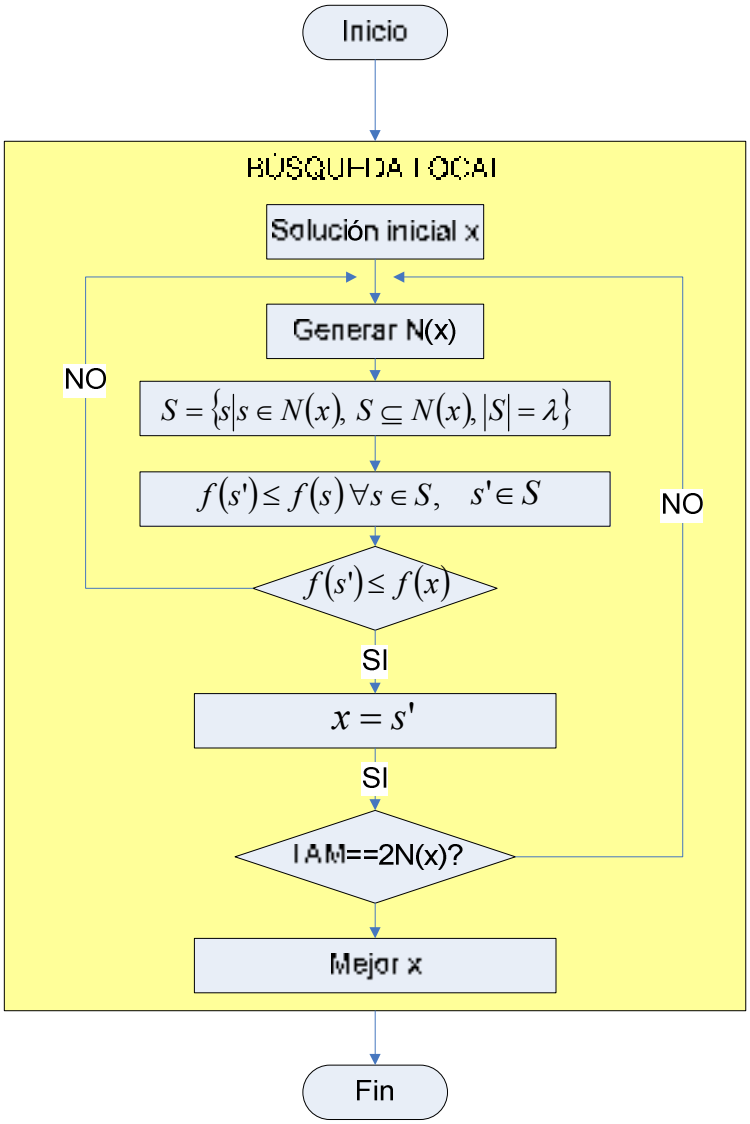

Figura 3.5. Algoritmo híbrido tipo Or-modificada

El mecanismo de vecindad aplicado al algoritmo genético se realizó a nivel de operador de mutación como lo muestra la Figura 3.6. La búsqueda local en vecindades tipo Or-modificada tiene un comportamiento similar al operador genético de mutación ya que se realizan intercambios entre nodos de un mismo individuo.

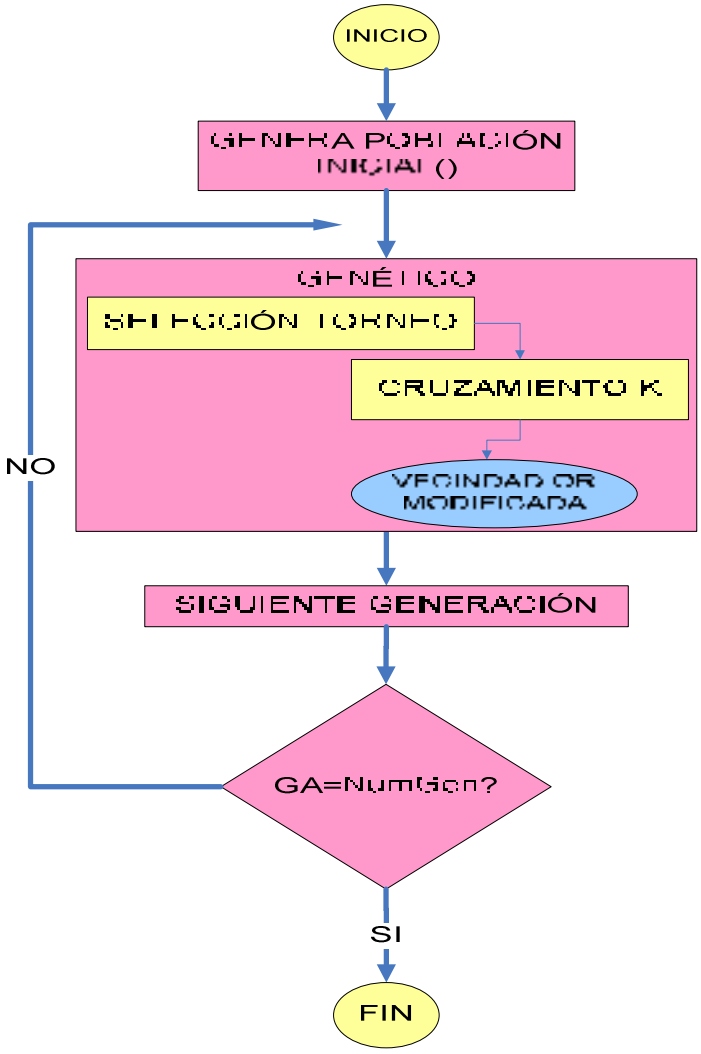

Figura 3.6. Algoritmo evolutivo secuencial con vecindad tipo Or-modificada

La combinación de operadores para la versión GA-VRPTW-SN son: la población inicial generada con la estrategia de agrupamiento k-means, la selección por torneo, el cruzamiento-K y una vecindad tipo OR modificada. La mayoría de éstos operadores ya han sido explicados en secciones anteriores a continuación se explicará el operador de selección por torneo. El operador de selección por torneo se toma del propuesto por Tan [Tan et al., 2001] que consiste básicamente en tomar la lista de la población (lista individuo P) y sacar una copia de esa lista y la copia se ordena de menor a mayor valor (lista individuo M), teniendo estas dos listas se procede a

comparar dos individuos de la lista individuo P y dos individuos de la lista individuo M, el que tenga menor valor de aptitud (F) de la lista individuo P se toma como candidato y se combina con el individuo de menor valor de aptitud (F) de la lista individuo M y así sucesivamente hasta el final de la lista como lo muestra la Figura 3.7.

Figura 3.7. Funcionamiento del operador de selección

En la siguiente sección se realiza el análisis de complejidad del algoritmo GA-VRPTW-SN.

3.2.3. Análisis de complejidad

Para encontrar la complejidad del algoritmo genético hibrido GA-VRPTW-SN en el peor de los casos se realiza el análisis explicado en el capítulo 1, sección 1.2. y con el código contenido en el Apéndice C de este documento.

En la ecuación 3.4 se muestra la ecuación temporal que representa al algoritmo GA-VRPTW-SN. El código vinvulado a este análisis se encuentra en el apéndice B.

$$T_{\text{sec}}(n) = c_1 + \sum_{i=0}^{n}(c_2 + \sum_{j=0}^{n}(c_3 + \sum_{k=0}^{n} c_4) + \sum_{l=0}^{n_1}(c_5 + \sum_{m=0}^{n} c_6) + \sum_{n=0}^{C_1}(c_7 + \sum_{o=0}^{n} c_8 + \sum_{p=0}^{C_2}(c_9 + \sum_{q=0}^{n} c_{10} + \sum_{r=0}^{n} c_{11} + \sum_{s=o}^{n} c_{12} + \sum_{t=0}^{n} c_{13} + \sum_{u=0}^{n} c_{14} + \sum_{v=0}^{n} c_{15}))) = \quad (3.1)$$

$$T_{\text{sec}}(n) = c_1 + c_2 n + c_3 n^2 + c_4 n^3 + c_5 n^2 + c_6 n^3 + C_1 c_7 n + C_1 c_8 n^2 + C_1 C_2 c_9 n^2 + C_1 C_2 c_{10} n^3 + C_1 C_2 c_{11} n^3 + C_1 C_2 c_{12} n^3 + C_1 C_2 c_{13} n^3 + C_1 C_2 c_{14} n^3 + C_1 C_2 c_{15} n^3))) = \quad (3.2)$$

$$T_{\text{sec}}(n) = c_1 + c_2 n + (c_3 + c_5 + c_8 + C_1 C_2 c_9) n^2 + (c_4 + c_6 + C_1 C_2 (c_{10} + c_{11} + c_{12} + c_{13} + c_{14} + c_{15})) n^3 = \quad (3.3)$$

$$T_{\text{sec}}(n) = c_1 + c_2 n + c_{16} n^2 + c_{17} n^3 \quad (3.4)$$

Donde *n* representa el tamaño de la instancia, C_1, representa el número de generaciones y C_2 representa el criterio de paro de la vecindad.

Se puede concluir que el polinomio que define la complejidad temporal del algoritmo genético secuencial, GA-VRPTW-SN en el peor de los casos es: $T(n) \in O(n^3)$.

Capítulo 4. Algoritmo evolutivo paralelo para el problema del transporte con ventanas de tiempo

Las heurísticas como métodos de aproximación a soluciones de problemas combinatorios han sido efectivas sin embargo los tiempos de computación asociados a la exploración del espacio de soluciones pueden ser muy grandes, por ello la computación paralela surge como una alternativa para mejorar el tiempo de búsqueda de soluciones [Holland, 1967] [Cruz y Moreno, 2007] [Crainic y Toulouse, 2003].

El proceso de transformación de un algoritmo secuencial en paralelo implica el paralelismo de datos (es cuando se asigna a varios procesadores elementos de datos y cada procesador realiza la misma operación simultáneamente sobre sus datos, es decir se ejecuta en SIMD o Simple Instrucción Múltiple Dato una misma secuencia de instrucciones sobre diferentes elementos de datos) y el paralelismo de control o funcional (es en el que se aplican diferentes operaciones sobre diferentes elementos de datos simultáneamente y se ejecutan sobre máquinas MIMD (Múltiple Instrucción Múltiple Dato). Para paralelizar hay que tomar en cuenta la granularidad del proceso (número de instrucciones secuenciales que forma el proceso). La granularidad gruesa se considera al proceso con muchas instrucciones, y granularidad fina se considera al proceso con menor número de instrucciones secuenciales.

Existen diferentes formas de diseñar un algoritmo paralelo, la que se utilizará en este trabajo de investigación es a partir de un algoritmo genético secuencial convertirlo a algoritmo genético paralelo explotando las partes que son paralelizables [Seyed, 1999] [Conway, 1963]. El algoritmo genético secuencial del que partimos es un algoritmo que da solución al problema de ruteo de vehículos con ventanas de tiempo.

Cantu-Paz [Cantu-Paz 1998] clasifica a los algoritmos genéticos paralelos en tres categorías: una la paralelización global (consiste en iterar el método de búsqueda orientado principalmente a reducir el tiempo de ejecución del método y no a lograr mayor exploración de soluciones), grano grueso y grano fino (están definidos de acuerdo al tamaño de la población).

Los pasos sugeridos por Cruz y Moreno [Cruz y Moreno, 2007] para paralelizar un algoritmo genético son: identificar los componentes en los cuales se puede obtener ganancia computacional significativa, por ejemplo; evaluación del vecindario de soluciones; realizar un esquema jerárquico con técnicas cooperativas multihilos de búsqueda, que mejoran no sólo la velocidad sino también la calidad de las soluciones, la eficiencia computacional y la robustez de la búsqueda; En este capítulo mostraremos una metodología general de conversión de un algoritmo genético secuencial a un algoritmo genético paralelo.

4.1. Metodología de construcción del algoritmo en paralelo

Existen trabajos realizados sobre algoritmos paralelos para diferentes tipos de problemas, con el fin de minimizar el tiempo de ejecución en la búsqueda de una solución. Sin embargo por la diversidad de problemas que existen y la variación entre cada uno de ellos es difícil encontrar un estándar o método para paralelizar problemas, existen análisis específicos al problema que se esté abordando, pero aun no existe una metodología en general. En este capítulo se propone una metodología general de construcción que contenga la conversión de una algoritmo secuencial a un algoritmo paralelo, independientemente del problema que se esté abordando, en la mayoría de los problemas existe básicamente: un modelo (definición del problema,

restricciones, variables implicadas), un método de solución (dependiendo del problema) y un tipo de algoritmo que mejor lo resuelve (determinístico, no determinístico). Para problemas de optimización combinatoria por ejemplo, el objetivo a conseguir es encontrar una solución a problemas difíciles de resolver, aplicando algoritmos no determinísticos que aproximen lo más posible a una solución optima en un tiempo de computación considerable, ésta actividad implica analizar los tiempos de ejecución del algoritmo propuesto y dependiendo de la instancia de entrada ver si es o no computable. La metodología de conversión propuesta consta de tres bloques principales: 1) la identificación del problema, 2) la técnica de solución y 3) el algoritmo, mostrados en la Figura 4.1

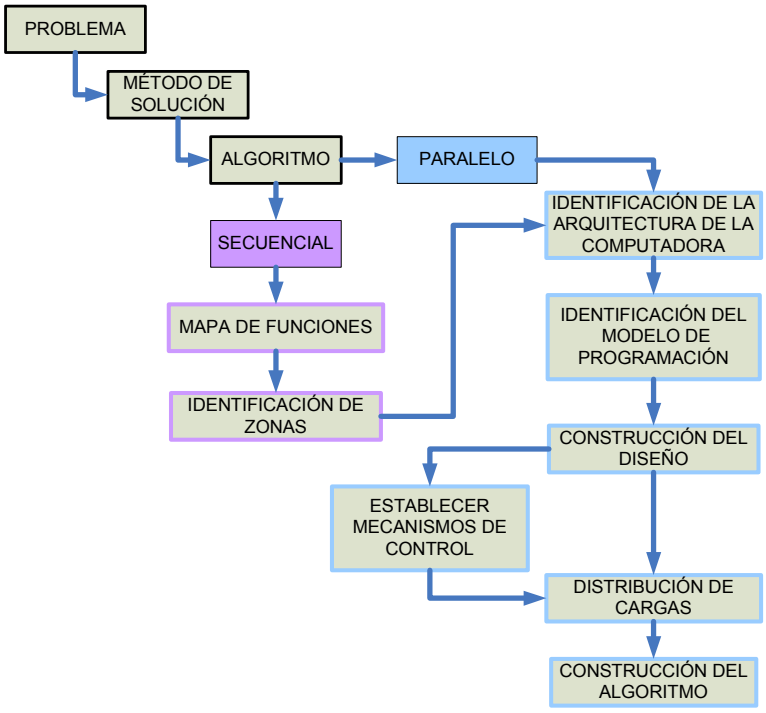

Figura 4.1. Metodología de construcción de algoritmo secuencial a paralelo

Descripción de los pasos de la metodología.

Paso 1. El bloque del problema se refiere al tipo de problema a resolver.

Paso 2. En el bloque de método de solución se refiere a analizar cómo construir la solución en base al tipo de problema que se esté analizando.

Paso 3. En el bloque de algoritmo se tienen tres opciones: primero, partiendo de un algoritmo secuencial construir su equivalente paralelo; segundo, construir una solución desde inicio con un algoritmo paralelo; tercero, mejorar un algoritmo paralelo existente.

Paso 4. Se analiza el algoritmo solución secuencial y se construye un mapa de funciones para tener clara la interacción de datos entre funciones dentro del algoritmo secuencial e identificar cuáles de las funciones implicadas en el algoritmo son dependientes e independientes; mediante la ejecución del algoritmo solución secuencial se establecen los tiempos de ejecución en cada una de las funciones sean dependientes o independientes.

Paso 5. Se realiza la identificación de zona que consiste en detectar las funciones con mayor tiempo de ejecución.

Paso 6. Una vez terminado el análisis del algoritmo secuencial, se examina la arquitectura de computadora con que se cuenta para la paralelización, según la taxonomía de Flynn [Flynn, 1972] [Flynn y Rudd, 1996] existen dos ramas principales: las computadoras basadas en número de procesadores y número de programas que se ejecutan y las computadoras en base a la estructura de la memoria. La primer rama se clasifica en cuatro tipos: SISD (Simple Instrucción Simple Dato), SIMD (Simple Instrucción Múltiple Dato), MISD (Múltiple Instrucción Simple Dato), MIMD (Múltiple Instrucción Múltiple Dato), la segunda rama se clasifica en dos tipos: memoria compartida y memoria distribuida; en memoria compartida se tienen dos tipos de acceso: el acceso uniforme a memoria es cuando se tienen procesadores idénticos

SMP (Simetric Multi Processing) y el acceso no uniforme cuando para cada procesador el acceso a la memoria es diferente; en memoria distribuida cada procesador solo tiene acceso a su propia memoria y la comunicación entre procesadores se hace por paso de mensajes MPI (Message Passing Interface).

Paso 7. El siguiente paso de la metodología es elegir el modelo de programación paralela a utilizar, existen básicamente cinco modelos: memoria compartida, hilos, MPI, paralelismo de datos e hibrido; dependiendo del equipo con que se cuente es el modelo de programación que se elige, en nuestro caso el modelo de programación que elegimos es el de memoria compartida e hilos combinada con paso de mensajes.

Paso 8. Posteriormente se realiza el diseño del algoritmo tomando en cuenta los principales ciclos que lo conformarán, es importante mencionar que si se parte de un algoritmo secuencial con complejidad polinomial, lo que se buscará en el algoritmo en paralelo es que la complejidad del algoritmo sea por ejemplo logarítmica.

Paso 9. Después de establecer el modelo de programación se establecerán mecanismos de control de acceso a la memoria solo en el caso de utilizar memoria compartida, los mecanismos de control o sincronización comúnmente utilizados en literatura son el de semáforos, secciones criticas o paso de mensajes.

Paso 10. Una vez analizados los mecanismos de control a utilizar, se procede a realizar la distribución de carga de cada proceso en procesadores o en hilos según sea el caso.

Paso 11. Ya que se tiene identificada cada una de las tareas se procede a construir el algoritmo lenguajes para programación paralela [Gill, 1958].

4.2. Diseño del algoritmo paralelo

Para poder diseñar el algoritmo en paralelo aplicaremos paso a paso la metodología descrita en la sección 4.1 al algoritmo genético secuencial para el problema de ruteo de vehículos con ventanas de tiempo.

Paso 1. Se selecciona el problema de optimización combinatoria VRPTW.

Paso 2. El método de solución es Heurística evolutiva.

Paso 3. Partiendo de un algoritmo secuencial construir su equivalente paralelo. Como la conversión es de un secuencial a un paralelo se debe descomponer en procesos. En la Figura 4.2 se muestra la descomposición del algoritmo secuencial en sus respectivas tareas principales con el fin de visualizar la comunicación entre datos dentro del algoritmo y establecer contadores de tiempo de ejecución en cada tarea, lo que detectará cual de las tareas toma mayor tiempo de ejecución que afecta la eficiencia del algoritmo en general.

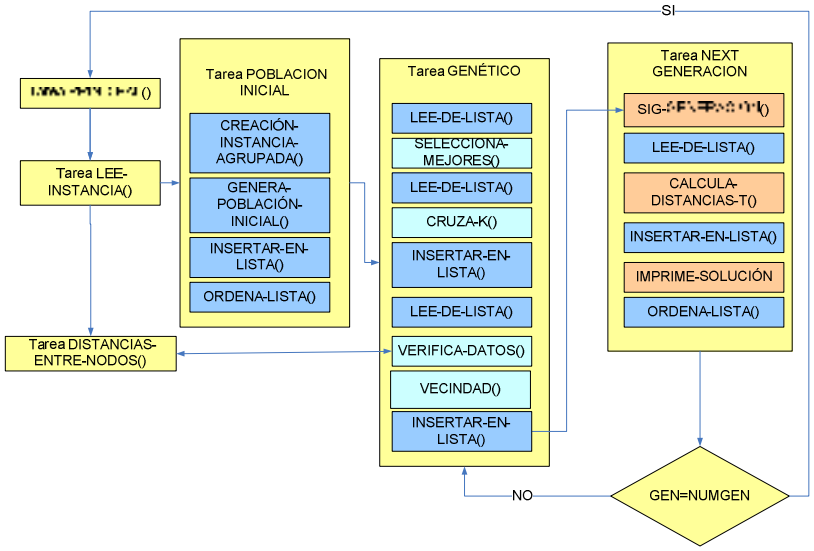

Figura 4.2. Descomposición del algoritmo en tareas principales

Paso 4. Después de detectar las tareas que consumen mayor tiempo de ejecución se realiza el mapa de funciones como lo muestra la Figura 4.3.

Paso 5. Se realiza la identificación de zonas que consiste en analizar los procesos de las tareas que consumen mayor tiempo y esos procesos son los candidatos a paralelizar o zonas identificadas. Como por ejemplo la función de vecindad que se representa en la Figura 4.3.

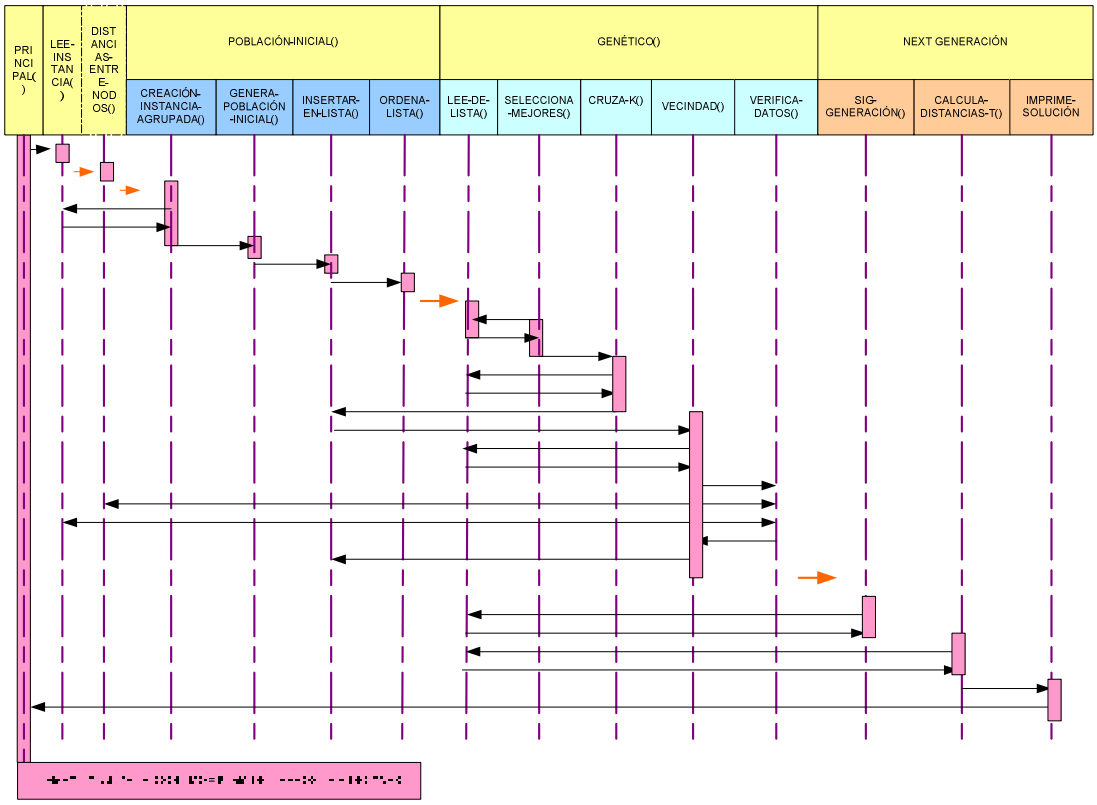

Figura 4.3. Mapa de funciones del algoritmo genético

Paso 6. Se analiza la arquitectura de la computadora en que se ejecutaran las pruebas, en nuestro caso se trabajará en un IBM PSeries 690 Supercomputadora Paralela IBM p690, 32 procesadores 1.3GHz y 32GB RAM. Este equipo soporta directivas basadas en memoria compartida como lo es Open MultiProcessing (OpenMP) y directivas basadas en memoria distribuida Message Passing Interface (MPI).

Paso 7. De acuerdo a la estructura del equipo, el modelo de programación para este caso puede ser un hibrido, es decir, en una parte del algoritmo trabajar con memoria compartida y en otros bloques trabajar con memoria distribuida.

Paso 8. El diseño del algoritmo en paralelo y su distribución de carga, dependerá mucho del estilo del programador y del ambiente de programación utilizado. Existen diferentes opciones de programación en paralelo, utilizando un lenguaje de programación especializado, utilizando un lenguaje de programación secuencial existente adaptándole constructores especiales para establecer el paralelismo o utilizar un lenguaje de programación secuencial existente más una librería de procedimientos para paralelismo. Algunos lenguajes de programación paralela son: OpenMP, HPF (High Performance Fortran), jade, Parallel APL, entre otros.

Paso 9. Como el modelo de programación puede ser hibrido cuando se trabaje con memoria compartida es necesario establecer mecanismos de control de acceso a memoria, según el gusto del programador mediante semáforos, sección crítica o paso de mensajes. En este caso se trabajará con secciones críticas utilizando el modelo de hilos de ejecución combinado con paso de mensajes.

Paso 10. Posteriormente se procede a la distribución de cargas, que puede ser por descomposición funcional o por descomposición de datos, en nuestro caso utilizamos procesos con hilos de ejecución por lo que la distribución de las cargas será por descomposición de datos. Como se muestra en la Figura 4.4.

Paso 11. Se realiza la construcción del algoritmo tomando en consideración todos los pasos anteriores.

El diseño del algoritmo en paralelo se muestra en la Figura 4.4

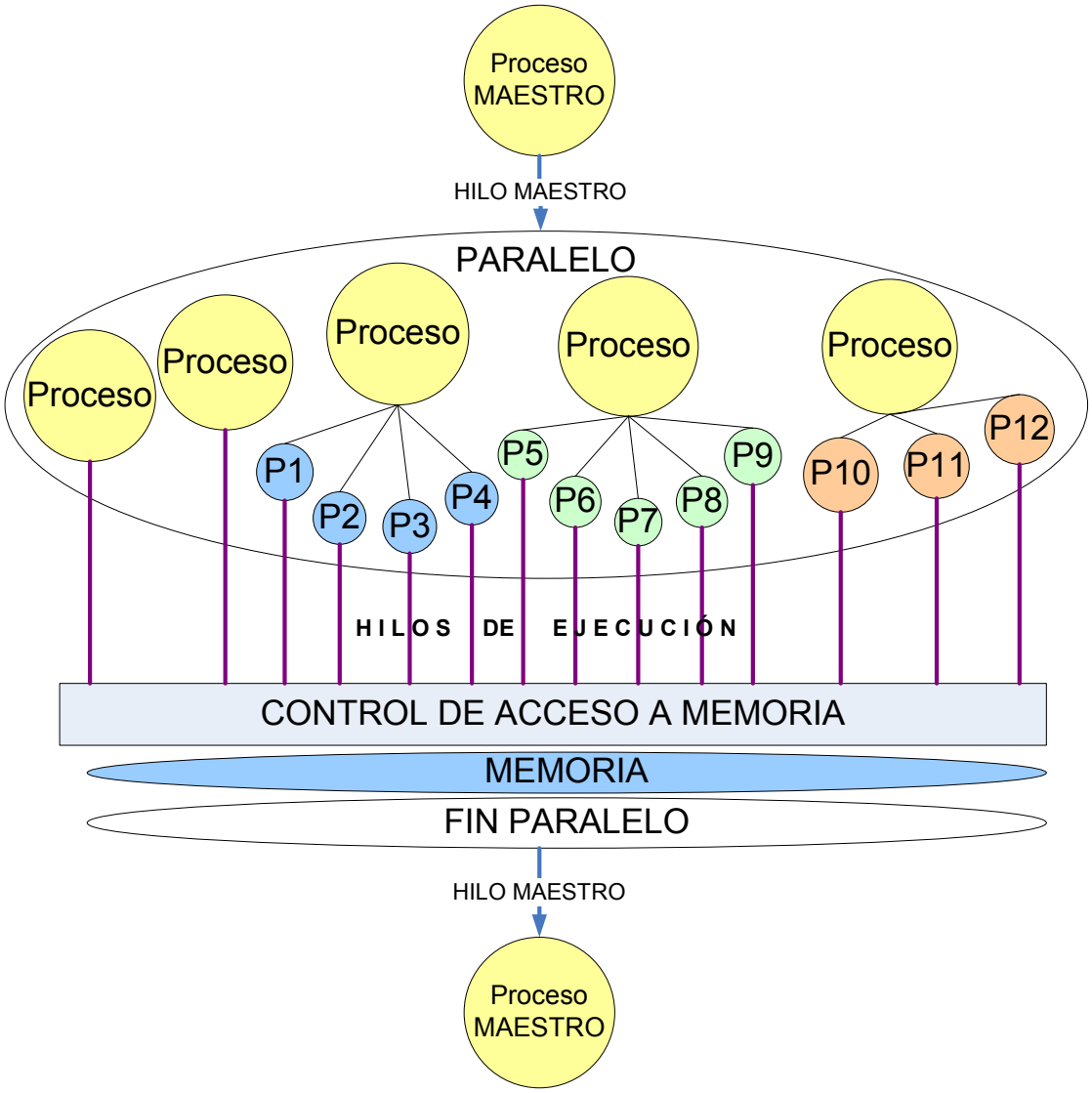

Figura 4.4. Diseño del algoritmo en paralelo

Donde el hilo maestro tiene el control de los procesos y a su vez los procesos tienen control sobre subprocesos mostrados en la Figura 4.4 como P1, P2 hasta P12. Tanto los procesos como subprocesos representan hilos de ejecución y todos comparten la misma memoria.

La aplicación de la metodología de conversión de algoritmos permitió detectar la zona problable a paralelizar dentro del algoritmo GA-VRPTW-SN. En la sección siguiente se explica el algoritmo evolutivo paralelizado.

4.3. Algoritmo evolutivo en paralelo

En la Figura 4.5 se ilustra el algoritmo evolutivo paralelizado. Donde la función correspondiente a la búsqueda local es la que se paralelizó, los procesos de generacion de población inicial, selección y cruzamiento se ejecutan de forma secuencial. Como todos los procesos comparten la memoria es necesario el control de acceso por medio de mecanismos de control de acceso como lo son semáforos, secciones criticas o paso de mensajes. Las directivas Open MP realizan de manera automática el control de acceso a memoria. Los operadores genéticos utlizados en GA-VRPTW-PN son los mismos utilizados por el GA-VRPTW-SN. Para la población inicial un mecanismo de agrupación k-means, un operador de selección torneo, un *cruzamiento-K* y una técnica de búsqueda local mediante la vecindad tipo Or-modificada. En la asignación de tareas por procesador cada vez que un procesador termina su tarea no espera a que los demás terminen sino que continúa con la siguiente tarea y asi sucesivamente para todos los procesadores de tal forma que las actividades de cada procesador son independientes lo que ocasiona un ahorro de tiempo en la ejecución total del algoritmo.

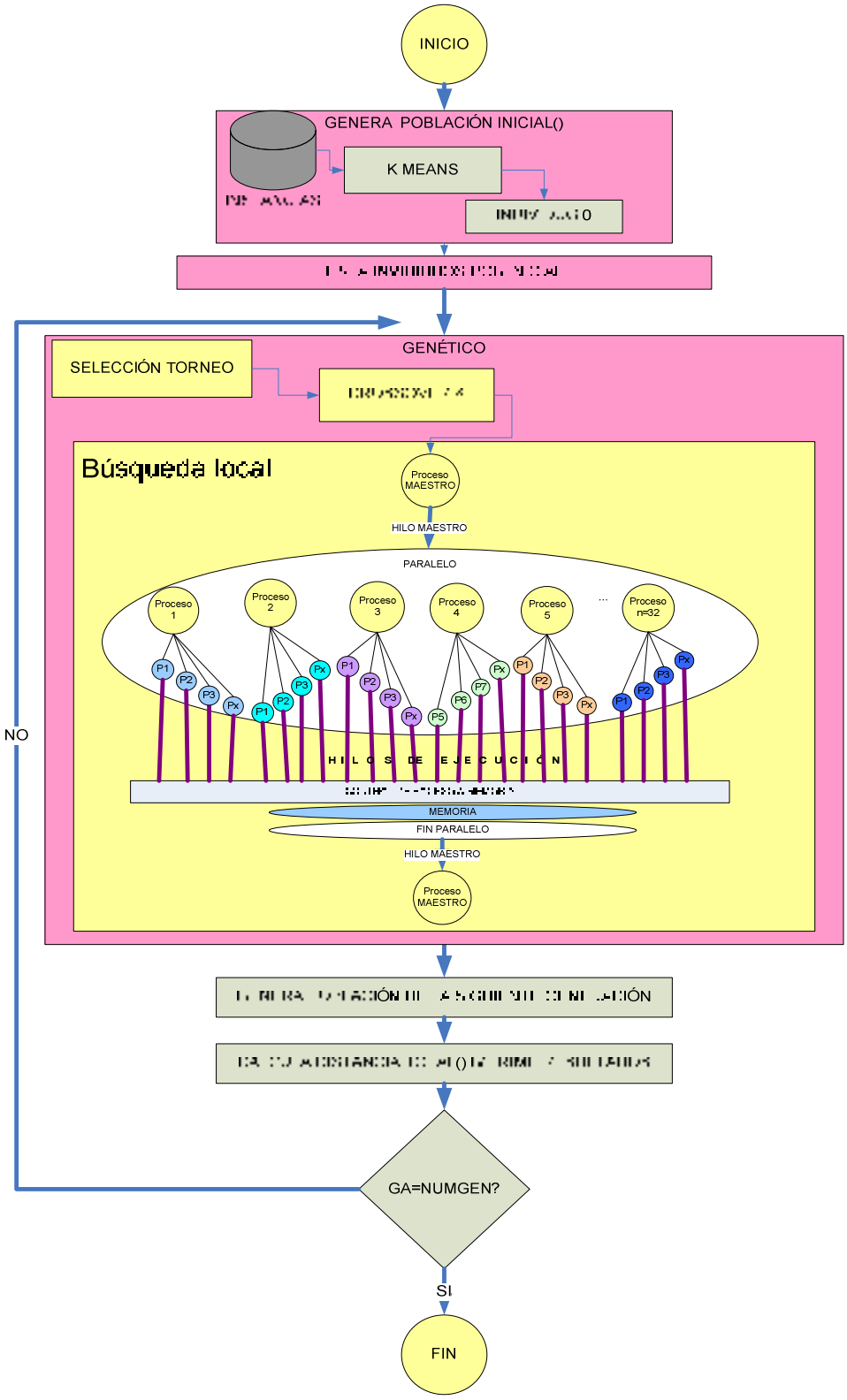

Figura 4.5 Algoritmo evolutivo paralelizado

4.3.1. Operadores

Los operadores genéticos utilizados para el algoritmo evolutivo en paralelo son el operador de selección por torneo y el cruzamiento-k descritos en el capítulo 3 en la sección 3.2.1. Para el caso de la mutación se aplicó búsqueda en vecindades descrita en la sección 3.2.2 y para la población inicial una técnica de agrupación k-means.

4.3.2. Análisis de complejidad

Uno de los objetivos por el cual buscan paralelizar, la mayoría de los autores que han trabajado con problemas combinatorios, es para optimizar el tiempo que tarda el algoritmo en encontrar una solución. La velocidad de un algoritmo en paralelo no depende del número de procesadores utilizados, sino de la granularidad de cada proceso. La distribución de las cargas por procesador se ilustra en la Figura 4.6 donde se muestra que la región paralela se lleva a cabo en el mecanismo de vecindad. Se pretende que cada individuo en la lista de población sea atendido por un hilo x en un procesador Y. De tal forma que a cada individuo se le aplique la función de genera vecinos totalmente independiente del siguiente individuo en la población. El bloque de instrucciones que se aplicara es el mismo en cada procesador solo cambia el dato de entrada (individuo). La siguiente generación de individuos se forma después de cada proceso de generación de vecinos haya concluido y almacene el mejor individuo en la lista de la población. Como no existen dependencias entre iteraciones de la función de vecindad; la paralelización realizada al algoritmo secuencial es a nivel de bucle. En donde el compilador asigna a cada procesador un subconjunto de

iteraciones de un bucle para su ejecución paralela. El compilador sólo puede aplicar esta técnica si está seguro de que ninguna iteración del bucle depende de otras, es decir, que todas las iteraciones pueden calcularse en cualquier orden.

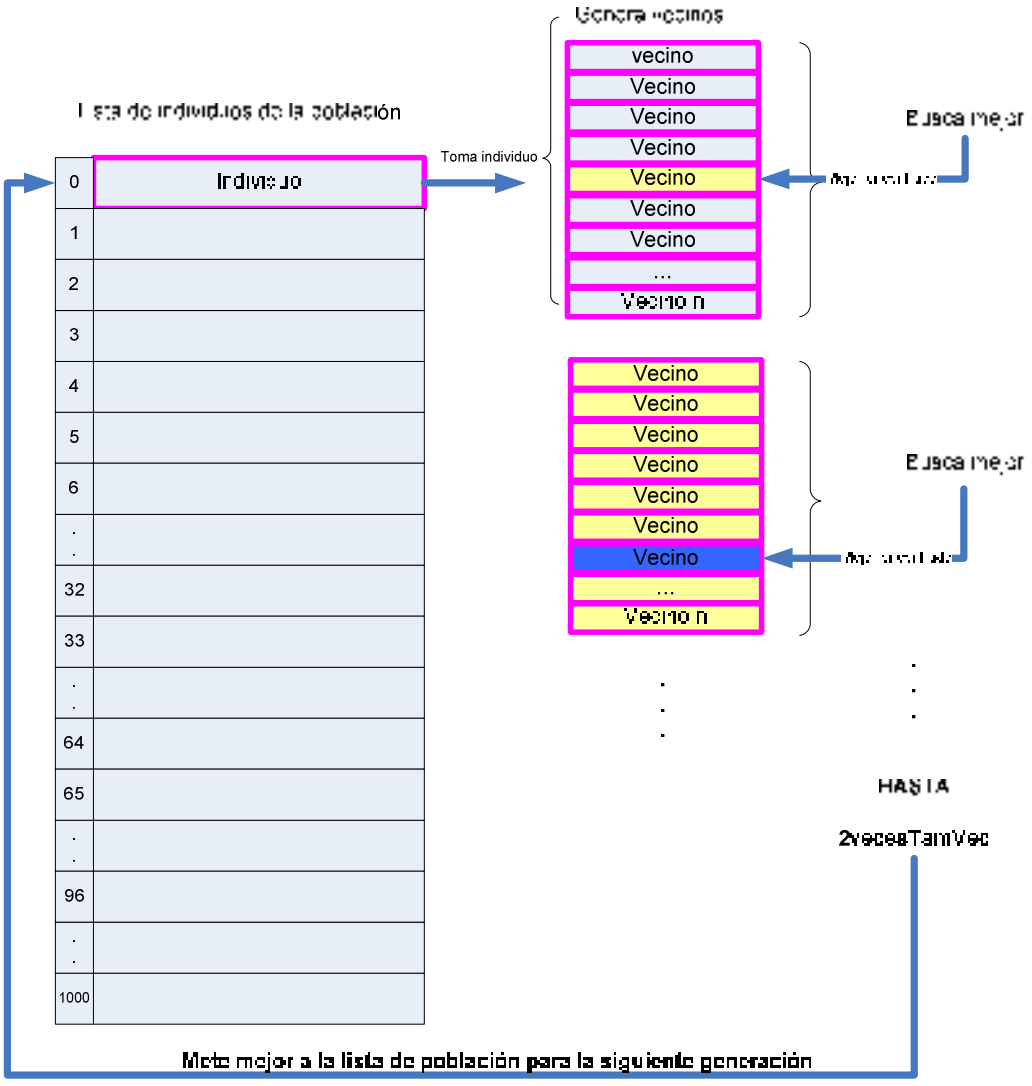

Figura 4.6. Distribución de cargas por procesador

En este caso para calcular los vecinos tomando como solución de inicio cada individuo de la población inicial no existe dependencia alguna de datos. Se pueden asignar tareas a diferentes hilos con el mismo código y diferentes

entradas para que empiecen a generar la población que conforma la siguiente generación. Cuando cada hilo haya concluido su tarea el sistema asignará a ese hilo el siguiente individuo en la lista de la población así sucesivamente hasta recorrer la lista que contiene a los mil individuos que forman la población.

De acuerdo a las reglas de asignación de análisis de algoritmos (Apéndice A) se obtiene el siguiente polinomio representado en la ecuación 4.4 que describe la complejidad temporal del algoritmo en paralelo. El código utilizado para el análisis se encuentra el el Apéndice C.

$$T_{Par}(n) = c_1 + \sum_{i=0}^{n}(c_2 + \sum_{j=0}^{n}(c_3 + \sum_{k=0}^{n} c_4) + \sum_{l=0}^{n_1}(c_5 + \sum_{m=0}^{n} c_6) + \sum_{n=0}^{C_1}(c_7 + \sum_{o=0}^{n})) = \quad (4.1)$$

$$T_{par}(n) = c_1 + c_2 n + c_3 n^2 + c_4 n^3 + c_5 n^2 + c_6 n^3 + C_1 c_7 n + C_1 n \quad (4.2)$$

$$T_{par}(n) = c_1 + (c_2 + C_1 c_7 + C_1)n + (c_3 + c_5)n^2 + (c_4 + c_6)n^3 \quad (4.3)$$

$$T_{par}(n) = c_1 + c_8 n + c_9 n^2 + c_{10} n^3 \quad (4.4)$$

La complejidad temporal del algoritmo paralelo en el peor de los casos es $T(n) \in O(n^3)$. El objetivo principal de la paralelización de un algoritmo no es reducir el grado de complejidad del mismo sino optimizar el tiempo que tarda en encontrar una solución.

Por el análisis de tiempos de ejecución presentado en la Figura 4.3 de cada función en el algoritmo secuencial GA-CRPTW-SN se detecto que la función de vecindades es la que implica más tiempo en ejecutarse y encontrar una solución por lo que la región paralela únicamente se aplicara a la vecindad para reducir el tiempo de ejecución del algoritmo.

El tiempo de procesamiento de la región paralela se analiza en base a la estructura empleada a nivel de bucle. El algoritmo propuesto distribuye a los individuos de la población original en *p* procesadores iguales de forma tal que cubre a todos los individuos de la población total, como se muestra en la ecuación (4.5).

$$\sum_{i=1}^{p} I_i = POB \tag{4.5}$$

Donde I_i representa el número de individuos de la población asignados al procesador p_i y POB representa los individuos totales de la población. De esta forma en cada procesador p_i se ejecuta la función de vecindario que a su vez se divide en hilos pertenecientes al procesador p_i.

El tiempo de proceso de la parte en paralelo aunado al tiempo de proceso de la parte secuencial genera la siguiente función temporal.

$$T_{paralelo} = g * TAMpob * (t_s + t_C + (NUMitera * \frac{(I * TAMvec * (t_{gv} + t_{bv})}{Numero\,\mathrm{Pr}\,ocesadores})) \tag{4.6}$$

$$T_{\mathrm{sec}} = g * TAMpob * (t_s + t_C + (NUMitera * (I * TAMvec * (t_{gv} + t_{bv}))) \tag{4.7}$$

Donde:

- $T_{paralelo}$ es el tiempo de proceso del algoritmo paralelo
- T_{sec} es el tiempo de proceso del algoritmo secuencial
- g es el número de generaciones
- TAMpob es el tamaño de la población
- ts tiempo de la selección
- tc tiempo del cruzamiento

- NUMitera el criterio de paro de la vecindad
- I el número de individuos asignados a uno o varios procesadores
- TAMvec el número de vecinos generados
- tgv el tiempo de generar el vecindario
- tbv el tiempo de búsqueda del mejor en el vecindario.

Analizando la ecuaciones (4.6) y (4.7) aparentemente los tiempos son iguales sin embargo la gran diferencia entre uno y otro radica en que en el proceso en paralelo se divide la tareas entre el número de procesadores lo que resulta en un tiempo de proceso menor. Si partimos de la expresión representada en la ecuación (4.8).

$$I_i = TAMpob / p \tag{4.8}$$

Entonces el tiempo de ejecución se reduce ya que se estará ejecutando un cierto número de individuos I por procesador y a su vez a cada individuo I asignado a un procesador *p* le será asignado un cierto número de hilos pertenecientes al procesador *p* para realizar la búsqueda en la vecindad generada por el individuo I .

La ganancia de tiempo del proceso en paralelo se ilustra en la ecuación (4.9).

$$\frac{T_{\text{sec}}}{T_{paralelo}} = \frac{g * TAMpob * (t_s + t_c + (NUMitera * (I * TAMvec * (t_{gv} + t_{bv}))))}{g * TAMpob * (t_s + t_c + (NUMitera * ((TAMpob / p) * TAMvec * (t_{gv} + t_{bv}))))}$$

$$\frac{T_{\text{sec}}}{T_{paralelo}} = \frac{I}{(TAMpob / p)} = \frac{I * p}{TAMpob} \tag{4.9}$$

De la ecuación (4.9) podemos observar que cuando se aumente el *TAMpob* el ahorro de tiempo será menor. Por lo que el proceso de paralelización del algoritmo genético secuencial si acelero el proceso de encontrar soluciones.

Capítulo 5. Experimentación y resultados

En este capítulo se explican las pruebas realizadas con las diferentes versiones del algoritmo y se realizan análisis de sensibilidad, estadístico y comparativo, y se da la descripción del equipo utilizado para su ejecución.

5.1. Descripción del equipo utilizado

Se utilizó el laboratorio[1] de optimización y software para las ejecuciones y pruebas con las diferentes versiones del algoritmo secuencial. El laboratorio se compone de 4 máquinas con las siguientes características: dos procesadores Pentium (R) M a 1.60 GHz, 1GB de memoria RAM, sistema operativo Windows XP y el compilador utilizado es Microsoft Visual C++ versión 6.0.

Las pruebas de paralelo se ejecutaron utilizando los 32 procesadores de la supercomputadora paralela IBM PSeries p690 Regatta- Aleph[2] La Regatta tiene las siguientes características: 32 procesadores, 1.3GHz y 32GB de memoria RAM. Este equipo soporta directivas basadas en memoria compartida como lo es OpenMP (Open MultiProcessing) y directivas basadas en memoria distribuida MPI (Message Passing Interface).

Las pruebas se realizaron para varias instancias (25 y 100 nodos) del benchmark de Solomon para el problema VRPTW. En las pruebas realizadas en regata se utilizaron los algoritmos GA-VRPTW-VS en versión secuencial y GA-VRPTW-VP en versión paralelo. Para el algoritmo secuencial GA-VRPTW-VS se realizó una sintonización de parámetros por

[1] El laboratorio de optimización y software está financiado por el proyecto 160 del fideicomiso SEP-UNAM 2006-2007

[2] La computadora Aleph está financiada a través del proyecto “Cómputo científico”. FOMES2000-SEP

medio de un análisis de sensibilidad que fue utilizado para ejecución de las pruebas con el algoritmo paralelo GA-VRPTW-VP. Las variables sintonizadas se muestran en el análisis de sensibilidad en la siguiente sección.

5.2. Análisis de sensibilidad

El análisis de sensibilidad es una forma de evaluación del comportamiento de ciertas variables dentro de un problema dado [Hillier y Lieberman 1991]. Originalmente el análisis de sensibilidad surge de la programación lineal (PL) como parte importante en la toma de decisiones; permite determinar cuándo una solución sigue siendo óptima, dados algunos cambios ya sea en el entorno del problema, en la empresa o en los datos del problema mismo.

El análisis de sensibilidad consiste en determinar qué tan sensible es la respuesta óptima del método Simplex, al cambio de algunos datos como las ganancias o costos unitarios (coeficientes de la función objetivo) o la disponibilidad de los recursos (términos independientes de las restricciones). La variación en los datos del problema se observan individualmente, es decir, se analiza la sensibilidad de la solución debido a la modificación de un dato a la vez, asumiendo que todos los demás permanecen sin alteración alguna.

El objetivo principal del análisis de sensibilidad es establecer un intervalo de números reales en el cual el dato que se analiza puede estar contenido, de tal manera que la solución sigue siendo óptima siempre que el dato pertenezca ha dicho intervalo [Hillier y Lieberman, 1991].

El análisis de sensibilidad se realiza para buscar la sintonización de variables de entrada al algoritmo. Para que el algoritmo mejore en eficiencia y eficacia se busca el porcentaje adecuado para las variables a sintonizar.

Las variables que comúnmente se observan son: los coeficientes de la función objetivo y los términos independientes de las restricciones.

Partiendo del modelo del problema VRPTW que consta de una función objetivo y una serie de ecuaciones que representan las restricciones del problema (como se describe en la sección 2.3.2) se realizó el análisis de sensibilidad obteniendo los parámetros de las variables sintonizadas.

La sintonización de parámetros se realiza para buscar los rangos que cada variable puede tomar para que el algoritmo arroje buenos resultados, utilizando cierto porcentaje de cada operador genético, de cruzamiento y de mutación.

Con motivo de encontrar el rango de variabilidad de cada parámetro se trabajaron varias versiones del algoritmo genético para el problema de VRPTW.

- La primera versión del algoritmo genético con la fijación de restricciones de precedencia (GA-PCP) [Cruz-Chávez et al., 2008] se trabajo con la variable de población.
- La segunda versión del algoritmo genético para el problema del transporte con ventanas de tiempo utilizando un operador de mutación inteligente (GA-VRPTW) [Díaz-Parra y Cruz-Chávez, 2008] se sintonizaron las variables de cruzamiento y mutación.
- En la tercera versión del algoritmo genético con vecindad (GA-VRPTW-SN) se trabajó con las variables del tamaño de la vecindad, muestras del tamaño de la vecindad y criterio de paro de la búsqueda en la vecindad.

- La cuarta versión (GA-VRPTW-PN) que se construyó fue la paralela del algoritmo secuencial con vecindad en esta cuarta y última versión los parámetros de sintonización fueron los mismos utilizados en la versión 3.

Las variables que se utilizaron para el análisis de sensibilidad en las diferentes versiones del algoritmo genético para el problema del ruteo de vehículos con ventanas de tiempo se enlistan a continuación:

- Tamaño de la población
- Porcentaje de mutación
- Porcentaje de cruzamiento
- Tamaño de la vecindad
- Tamaño de la muestra
- Criterio de paro de la vecindad

Las variables sintonizadas para cada una de las versiones del algoritmo propuesto como solución al problema ruteo de vehículos con ventanas de tiempo se describen a continuación.

Para la versión uno del algoritmo GA-PCP. Las variables con el parámetro de sintonización adecuado y operador de selección, cruzamiento y mutación se registran en la Tabla 5.1; donde las variables de selección, cruzamiento y mutación permanecen constantes. La variación de los parámetros se realizó con la variable de población usada en la investigación [Cruz-Chávez y Díaz-Parra, 2008]. Los tipos de operadores genéticos utilizados en el algoritmo versión 1 GA-PCP, fueron seleccionados de las diferentes combinaciones realizadas con los operadores genéticos mostrada en la Tabla 3.1, las

combinaciones se realizaron entre el operador de selección y cruzamiento, el operador de mutación permaneció constante.

Tabla 5.1. Parámetros de sintonización de variables para el algoritmo genético con PCP y operadores genéticos de selección, cruzamiento y mutación

ALGORITMO	Variables de sintonización							
	Población individuo factible 1000 individuos 5 generaciones		Selección		Cruzamiento		Mutación	
	%	Tipo	%	Tipo	%	Tipo	%	Tipo
GA-PCP	30	Población	100	Torneo	0.1	Cruzamiento-k	0.5	Flip-bit
	70	PCP						

Con base en el análisis realizado con la variación de poblaciones se detectó que para un porcentaje mayor de población generada con la técnica de PCP el algoritmo reporta soluciones cercanas al óptimo reportado en literatura para instancias del benchmark de Solomon de 25 nodos para el problema VRPTW. Para el caso de las instancias de 100 nodos con esta combinación la sintonización de variables no obtuvo buenos resultados.

Para la versión 2 del algoritmo GA-VRPTW las variables con el parámetro de sintonización adecuado para obtener resultados óptimos para el algoritmo genético con operador de selección, cruzamiento y mutación se registran en la Tabla 5.2. En la versión 2 se propone generar la población inicial sólo con individuos factibles sin aplicar PCP y con otro tipo de operador o técnica de mutación. El operador de mutación que se propone es usando una mutación no aleatoria condicionada a las restricciones propias del problema de transporte con ventanas de tiempo. Esta mutación se propone debido a la necesidad de mantener soluciones factibles en cada generación del algoritmo GA-VRPTW en contraste con los operadores de mutación

encontrados en la literatura que realizan la permutación generando números aleatorios; al aplicar este criterio de aleatoriedad al algoritmo GA-VRPTW la solución del problema VRPTW se vuelve infactible en la mayoría de los casos.

Tabla 5.2 Parámetros de sintonización de variables para el algoritmo genético con operadores genéticos de selección, cruzamiento y mutación

ALGORITMO	Variables de sintonización						
	Población 1000 individuos 20generaciones	Selección		Cruzamiento		Mutación	
	%Población	%	Tipo	%	Tipo	%	Tipo
GA-VRPTW	100 factibles	100	Torneo	0.1	C-k	0.6	M-S

Donde C-k indica el operador de cruzamiento-k, M-S indica el operador de mutación-S.

Para la versión 3 del algoritmo GA-VRPTW-SN se usó la sintonización de parámetros que se muestra en la Tabla 5.3, para este caso los parámetros de sintonización de selección y cruzamiento se obtuvieron de la sintonización anterior mostrada en la Tabla 5.2. Para la población inicial se utilizó la técnica de k-means y en el operador de mutación la técnica de búsqueda local en vecindades a nivel de operador de mutación. Las pruebas de sintonización de parámetros que se realizaron para el algoritmo GA-VRPTW-SN fueron relacionados al mecanismo de vecindad. La sintonización de las variables de vecindad se realizó partiendo de un tamaño de muestra de vecindad fijo de 10 y variando el criterio de paro de la búsqueda iniciando con diez e incrementando hasta llegar a 500, posteriormente se fijo el criterio de paro de la búsqueda y se hicieron

variaciones del tamaño de la muestra en la vecindad, se inicio con un tamaño de diez individuos incrementando hasta llegar a 500.

Tabla 5.3 Parámetros de sintonización de variables para el algoritmo genético con vecindad tipo Or modificada

ALGORITMO	Variables de sintonización								
	Pob=1000 g=20		Selección		Cruzamiento		Vecindad		
	%	Tipo	%	Tipo	%	Tipo	% Tamaño Muestra	Criterio Búsqueda 2TAM=100 %	Tipo
GA- VRPTW-SN	100	k-m	100	Torneo	0.01	C-k	10	10	Or-m

Donde *Pob* se refiere a la población formada por 1000 individuos, *g* es el número de generaciones, *k-m* el mecanismo de agrupación *k-means.* C-k es el cruzamiento-k, Or-m se refiere a la vecindad tipo Or modificada.

Para la versión 4 el algoritmo GA-VRPTW-PN la sintonización aplicada se obtuvo en el análisis de la versión 3 mostraba en la Tabla 5.4 esta sintonización se utilizó en el algoritmo genético paralelo. Para la versión del algoritmo en paralelo se realizo también la sintonización en base al tamaño de la muestra de vecinos y criterio de paro de búsqueda en la vecindad.

Tabla 5.4 Parámetros de sintonización de variables para el algoritmo genético con vecindad tipo Or modificada en paralelo

ALGORITMO	Variables de sintonización								
	Población 1000 individuos 20 generaciones		Selección		Cruzamiento		Vecindad		
	%	Tipo	%	Tipo	%	Tipo	% Tamaño Muestra	Criterio Búsqueda 2TAM=100 %	Tipo
GA-VRPTW-PN	100	k-means	100	Torneo	0.01	C-k	10	10	Or-m

C-k indica el operador de cruzamiento-k, Or-m indica el operador de mutación-S.

En la siguiente sección se muestra el análisis de resultados obtenidos con las diferentes versiones del algoritmo que se propone como solución al problema de ruteo de vehículos con ventanas de tiempo.

5.3. Análisis estadístico

En problemas combinatorios el análisis estadístico de datos es de suma importancia para visualizar el comportamiento del problema en base a un algoritmo. En el análisis estadístico se estudian los resultados obtenidos del algoritmo por medio de los parámetros usando la media, desviación estándar y error relativo. En la media se obtiene el promedio de la solución de varias ejecuciones lo que permite saber si el algoritmo está dando resultados cercanos a los esperados. La desviación estándar permite saber que tanto están dispersas unas de otras las soluciones proporcionadas por el algoritmo. El error relativo permite saber que tan cercano esta el resultado arrojado por el algoritmo con respecto a una cota esperada. El análisis estadístico permite la toma de decisión en cuanto a que si un algoritmo es mejor o peor solución a un problema.

A continuación se muestra el análisis estadístico correspondiente a cada una de las versiones del algoritmo.

Para la primera versión (GA-PCP) se realizaron 100 ejecuciones por cada una de las instancias. Las instancias de prueba utilizadas en esta versión son sólo para las que se reportan en la literatura con un resultado óptimo con el fin de tener un rango de comparación con el algoritmo GA-PCP. El

tamaño de las instancias seleccionadas del benchmark de Solomon es de 25 nodos. El análisis estadístico se reporta en la Tabla 5.5.

Tabla 5.5 Análisis estadístico del algoritmo GA-PCP

Instancia	V	Major	peor	Promedio	σ	Op*	V*	ER
C104-25	2	186.9	257.3	189.8	4.05	186.9	3	0
R104-25	3	436.3	559.7	467.8	21.48	416.9	4	4.60
RC108-25	5	294.7	382.7	300.4	15.03	294.5	3	0.08
C204-25	2	286.0	367.9	290.9	4.51	213.1	1	34.25
R208-25	2	329.1	456.8	332.0	3.01	328.2	1	0.30
RC208-25	2	271.8	300.6	285.9	10.17	269.1	2	1.01

Donde *V* representa el número de vehículos encontrados en la solución, *mejor* representa la mejor solución encontrada, *peor* la pésima solución, *Op** representa el mejor valor reportado en literatura para cada instancia, *V** el mejor número de vehículos reportado en literatura y *ER* el error relativo. Cabe mencionar que el cálculo del error relativo utilizado en este documento es en base a la razón de la mejor solución encontrada contra el mejor reportado en literatura, tal razón se representa en la ecuación (5.1).

$$ER = \frac{mejor - Op*}{Op*} \tag{5.1}$$

Para la segunda versión (GA-VRPTW) los resultados obtenidos por el algoritmo y los óptimos reportados en literatura se muestran en la Tabla 5.6. Nótese que para las instancias C101, C105 y C108, el algoritmo GA-VRPTW llego al mejor reportado en literatura y para las demás instancias está cercano al mejor reportado con un error relativo pequeño.

Tabla 5.6. Resultados obtenidos con el algoritmo genético con mutación inteligente

Instancia	V	Mayor	Media	σ	Op*	V*	ER
c101-100	10	**827.3985**	962.7439	107.8829	**827.3**	10	0.0119062
c102-100	10	828.2443	1027.1140	126.6793	827.3	10	0.1141423
c103-100	10	827.4851	986.2039	116.9508	826.3	10	0.1434304
c104-100	10	824.0308	1077.0705	124.9379	822.9	10	0.1374278
c105-100	10	**827.3167**	950.8616	99.4707	**827.3**	10	0.0020294
c116-100	10	827.4854	970.5487	109.2358	827.3	10	0.0224102
c107-100	10	827.8844	992.7809	122.4051	827.3	10	0.0706498
c108-100	10	**827.3336**	962.8849	105.4526	**827.3**	10	0.0040704
c109-100	10	827.6744	970.1983	106.5438	827.3	10	0.0452577

Para la versión 3 (GA-VRPTW-SN) se realizaron ejecuciones en el laboratorio de optimización y software, y otras ejecuciones en Aleph para la comparación de resultados con la versión en paralelo. En la Tabla 5.7 se muestran los resultados obtenidos en el laboratorio de Optimización y Software. Donde se observa que para las instancias C101-C103 y C105, C106 y C108 el error relativo es muy pequeño lo que indica que está muy cerca del mejor reportado y con respecto al número de vehículos el algoritmo GA-VRPTW-SN dio el mismo número de 10 vehículos como lo reportan en literatura. El número de ejecuciones para cada instancia fue de 30 ejecuciones con 20 generaciones.

Tabla 5.7. Resultados obtenidos del algoritmo genético con vecindad en su versión secuencial experimentados en laboratorio de optimización y software

Instancia	V	Mejor	Media	σ	Op*	V*	ER
c101-100	10	**827.321**	828.804	2.825	**827.3**	10	0.00002
c102-100	10	**827.378**	829.608	2.967	**827.3**	10	0.00009
c103-100	10	**826.343**	826.447	0.118	**826.3**	10	0.00005
c104-100	10	823.090	824.989	1.571	822.9	10	0.00020
c105-100	10	**827.318**	827.616	0.595	**827.3**	10	0.00002
c106-100	10	**827.301**	827.458	0.164	**827.3**	10	0.000001
c107-100					827.3	10	
c108-100	10	**827.338**	827.865	0.930	**827.3**	10	0.00004
c109-100					827.3	10	

Los resultados obtenidos del algoritmo GA-VRPTW-SN en Aleph con la misma sintonización que las ejecuciones en el laboratorio de optimización y software se muestran en la Tabla 5.8. El número de ejecuciones para cada instancia fue de 10 con 20 generaciones. Se observa que los resultados para las instancias C101, C102, C103, C104, C105, C106 y C109 están muy cercanos al mejor reportado en literatura. Cabe mencionar que el número de ejecuciones se redujo a 10 por la disponibilidad de la computadora Aleph.

Tabla 5.8. Resultados obtenidos del algoritmo genético con vecindad en su versión secuencial ejecutado en la computadora Aleph

Instancia	V	Mejor	Media	σ	Op*	V*	ER
c101-100	10	**827.338**	827.505	0.081	**827.3**	10	0.00004
c102-100	10	**827.307**	827.562	0.225	**827.**3	10	0.00031
c103-100	10	**826.320**	826.395	0.055	**826.3**	10	0.00011
c104-100	10	**822.992**			**822.9**	10	
c105-100	10	**827.305**	827.523	0.191	**827.3**	10	0.000006
c116-100	10	**827.307**	828.310	5.202	**827.3**	10	0.000008
c107-100					827.3	10	
c108-100	10	**827.313**	827.422	0.078	**827.3**	10	0.00014
c109-100					827.3	10	

Para la versión 4 (GA-VRPTW-PN) los resultados que se muestran en la Tabla 5.9. Las ejecuciones se realizaron en la computadora Aleph utilizando los 32 procesadores. El número de ejecuciones realizadas fue de 10 por cada instancia. Los resultados muestran que en la mayoría de los valores obtenidos para el recorrido total de las rutas el valor del error relativo fue muy pequeño y para las instancias C102, C105 y C107 prácticamente fue de cero. El número de vehículos utilizados para los recorridos en cada una de las instancias fue de 10 vehículos como los que reportan los mejores resultados.

Tabla 5.9. Resultados obtenidos del algoritmo genético con vecindad en su versión paralela ejecutado en la computadora Aleph

Instancia	V	Mejor	Media	σ	Op*	V*	ER
c101-100	10	**827.301**	827.393	0.084	**827.3**	10	0.000001
c102-100	10	**827.300**	827.323	0.005	**827.3**	10	0.000000
c103-100	10	**826.303**	826.310	0.015	**826.3**	10	0.000003
c104-100	10	**822.905**			**822.9**	10	
c105-100	10	**827.300**	827.341	0.003	**827.3**	10	0.000000
c116-100	10	**827.305**	827.353	0.005	**827.3**	10	0.000006
c107-100	10	**827.300**	827.318	0.009	**827.3**	10	0.000000
c108-100	10	**827.316**	827.342	0.032	**827.3**	10	0.000019
c109-100	10				**827.3**	10	

En la siguiente sección se muestra el análisis comparativo del algoritmo propuesto contra otras soluciones realizadas por otros autores para el problema del transporte con ventanas de tiempo.

5.4. Análisis comparativo

Proponer soluciones para problemas difíciles de resolver es complicado por la cantidad de restricciones del problema y por el crecimiento del espacio de soluciones para instancias grandes. Para saber si una propuesta de solución a este tipo de problemas es aceptable es necesario un análisis comparativo con otras propuestas de solución. La comparación debe ser con soluciones propuestas similares a la que se desea evaluar. En este caso el análisis comparativo se realizará con soluciones propuestas por otros autores que

han trabajado con algoritmos genéticos. La comparación de la propuesta de solución permitirá saber si la propuesta de solución es aceptable.

Algunos de los resultados de algoritmos genéticos utilizados en la comparación de las diferentes versiones del algoritmo propuesto, se han tomado del sitio web para VRP (http://neo.lcc.uma.es/radi-aeb/WebVRP/. En este sitio se muestran los mejores trabajos realizados por diferentes autores para el problema del transporte en sus diferentes variantes) [Dorronsoro-Díaz, 2007]. Otros resultados de algoritmos se han tomado del laboratorio de heurísticas [Wagner y Affenzeller, 2004]. Y otros más de diferentes autores que han trabajado con algoritmos genéticos para el problema del transporte con ventanas de tiempo.

En la sección 5.4.1 se muestran las tablas de resultados comparativos con otros algoritmos secuenciales genéticos y en la sección 5.4.2 se muestran las tablas de resultados comparativos solo de la versión paralela con otros algoritmos en paralelo.

5.4.1. Análisis comparativo con algoritmos secuenciales

Los algoritmos de comparación que se utilizaron para la versión GA-PCP y la versión GA-VRPTW fueron tomados del laboratorio de heurísticas tales algoritmos son: GGA (Generic Genetic Algorithm) [Affenzeller, 2002], SSGA (Steady-State Genetic Algorithm) [Chafekar, et al., 2003] y SXGA (Sexual Genetic Algorithm) [Wagner y Affenzeller, 2005]. Para la versión 3 (GA-VRPTW-SN) los algoritmos utilizados para la comparación fueron GenSAT [Thangiah, 1999], GIDEON [Thangiah, 1991] y GENEROUS [Potvin y Bengio, 1996]. El algoritmo GA-VRPTW-PN se comparó en desempeño con el algoritmo BBB (Berger Barkaoui Bräysy) [Berger et al., 2004] que es un

algoritmo genético paralelo para el problema de transporte con ventanas de tiempo.

Para la versión1 (GA-PCP) los resultados obtenidos se comparan con otros algoritmos genéticos para el problema del transporte y se presentan en la Tabla 5.10 Las condiciones de ejecución fueron las mismas para todos los algoritmos, el número de generaciones fue de 5 y el tamaño de la población fue de 1000 individuos.

Tabla 5.10. Resultados comparativos de eficiencia y eficacia de GA-PCP con otros algoritmos genéticos que aplican la técnica de satisfacción de restricciones

Benchmark	GA-PCP		GGA		SSGA		SXGA		Op
	UB	t, sec	UB	t, sec	UB	t, sec	UB	t, sec	
C104.25	**186.9**	42.2	189.3	78.8	187.7	59.4	189.7	79.0	**186.9**
R104.25	430.0	47.8	**416.9**	79.0	417.6	0.9	418.0	79.6	**416.9**
RC108.25	294.6	49.9	295.1	79.0	294.9	0.7	295.4	78.9	294.5
C204.25	279.0	47.9	221.1	79.8	223.3	0.9	223.3	79.7	213.1
R208.25	329.1	49.1	329.1	78.6	329.1	0.7	329.1	79.2	328.2
RC208.25	270.1	52.9	270.6	87.8	269.3	0.8	270.0	87.3	269.1

Los resultados obtenidos muestran que el comportamiento del GA-PCP para la instancia C104-25 llego al mejor valor reportado en literatura para las demás instancias de prueba se observa que los valores obtenidos están muy cercanos a los valores reportados en literatura. En lo que respecta a los tiempos de ejecución el algoritmo GA-PCP resultó ser mejor que los algoritmos GGA, SSGA y SXGA. El comportamiento del algoritmo con respecto al número de generaciones y el recorrido total para la instancia C104-25 se muestra en la Figura 5.1.

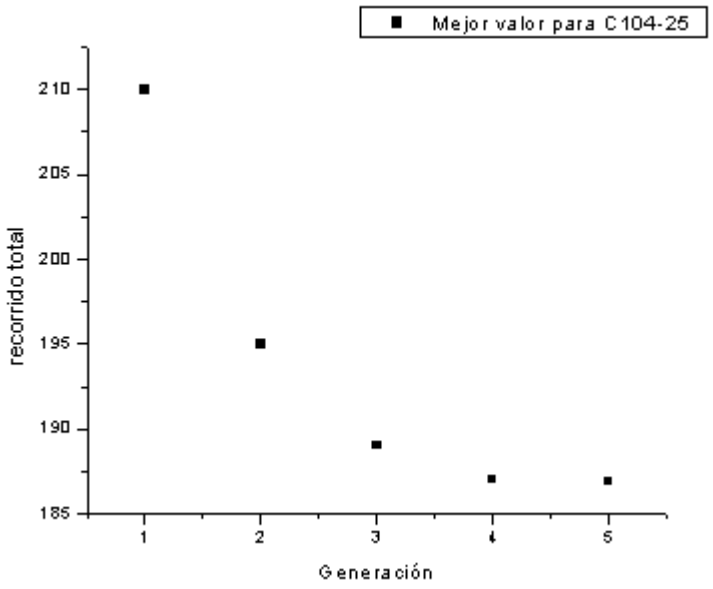

Figura 5.1. Resultados de GA-PCP para instancia C104-25

Como se observa en la Figura 5.1 los resultados obtenidos por el algoritmo GA-PCP convergen desde la generación 4 y en la generación 5 se obtiene el mejor valor de los reportados en literatura.

Para la versión 2 (GA-VRPTW) los resultados comparados con algoritmos del laboratorio de heurísticas [Wagner y Affenzeller, 2004] que utilizan algoritmos genéticos se presentan en la Tabla 5.1. Los parámetros de entrada para las pruebas se realizaron con una población inicial de 1000 individuos, para 20 generaciones. Las instancias utilizadas para la experimentación fueron las instancias con características de agrupación C definidas por Solomon para 100 nodos. El algoritmo genético propuesto GA-VRPTW, compite en eficacia y en eficiencia con los algoritmos GGA y SXGA. La estrategia de agrupación k-means aplicada para la construcción de la población inicial en combinación con la mutación muta-S genero buenos resultados para instancias que por definición están representadas en el espacio euclidiano de forma arracimada, como es el caso de las Instancias C.

Tabla 5.11.Resultados comparativos de eficiencia y eficacia de GA-VRPTW contra otros algoritmos genéticos para VRPTW

Instancia	**GA-VRPTW**		GGA		SSGA		SXGA		Op*
	Mejor	t, sec	Mejor	t, sec	Mejor	t, sec	Mejor	t, sec	
C101-100	**827.3**	2223	931.0	7221	1099.9	23	1143.7	2100	827.3
C102-100	828.2	2734	1159.1	1740	1204.2	24	1159.8	2040	827.3
C103-100	827.4	2196	1204.8	1680	1184.8	24	1222.6	2220	826.3
C104-100	824.0	2515	1109.9	1740	1256.6	22	1101.9	1920	822.9
C105-100	**827.3**	2313	1047.1	2100	1038.1	24	938.9	1860	827.3
C106-100	827.4	2131	1108.0	2280	1163.4	25	1046.2	2160	827.3
C107-100	827.8	2048	1171.9	3180	1301.8	23	1064.8	1800	827.3
C108-100	**827.3**	2051	1008.9	1860	1224.7	24	1072.1	2220	827.3
C109-100	827.6	1985	1143.8	1800	1208.3	24	1164.7	2460	827.3

Por los resultados obtenidos en la Tabla 5.11 se observa que el algoritmo GA-VRPTW para las instancias C101, C105 y C108 llego al mejor reportados en literatura (Op*); para las demás instancias genera valores cercanos al mejor reportado. La eficacia del algoritmo GA-VRPTW es alta comparada con los algoritmos GGA (general genetic algorithm), SSGA (steady state genetic algorithm) y SXGA (sexual genetic algorithm) del laboratorio de heurísticas; porque GA-VRPTW genera resultados cercanos al mejor reportado y para las instancias C101, C105 y C108 llega al mejor resultados. En cuanto a la eficiencia los algoritmos del laboratorio de heurísticas son un poco más rápidos que el algoritmo GA-VRPTW, esto puede ser debido a que el GA-VRPTW utiliza el mecanismo de mutación "*mutación-S*" que realiza un número mayor de operaciones al tratar de minimizar la distancia total recorrida por medio de varias búsquedas entre genes del individuo, lo que le consume tiempo al algoritmo para encontrar

una solución; pero para algunos casos como el del algoritmo GGA para la instancia C101, C106, y C107, el algoritmo SXGA para la instancia C103 y C106 los tiempos de GA-VRPTW son mejores.

La gráfica de los resultados por generación de las instancias C101, C105 y C108 para las cuales el algoritmo GA-VRPTW generó buenos resultados se presentan en las Figuras 5.2 a 5.4 que se muestran a continuación.

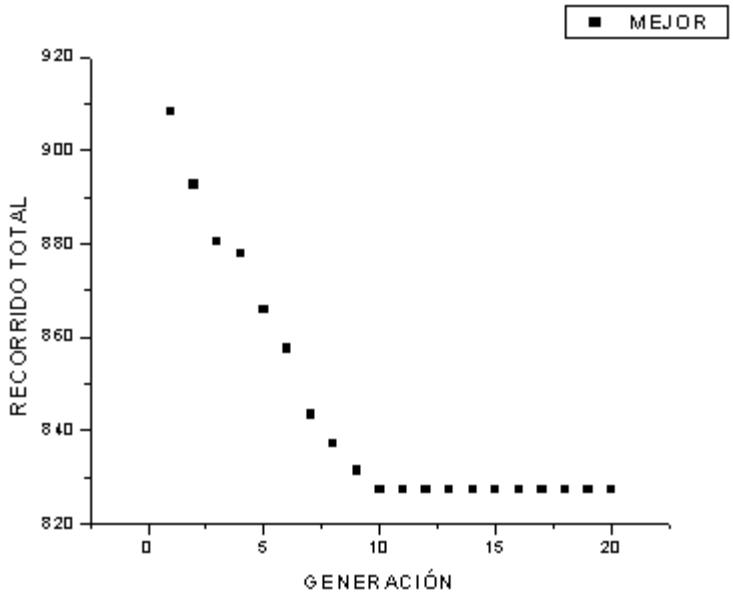

Figura 5.2. Resultados de GA-VRPTW por generación para instancia C101

Para la instancia C101 la convergencia al mejor valor se presentó en la generación 10 a partir de esa generación los valores se repitieron.

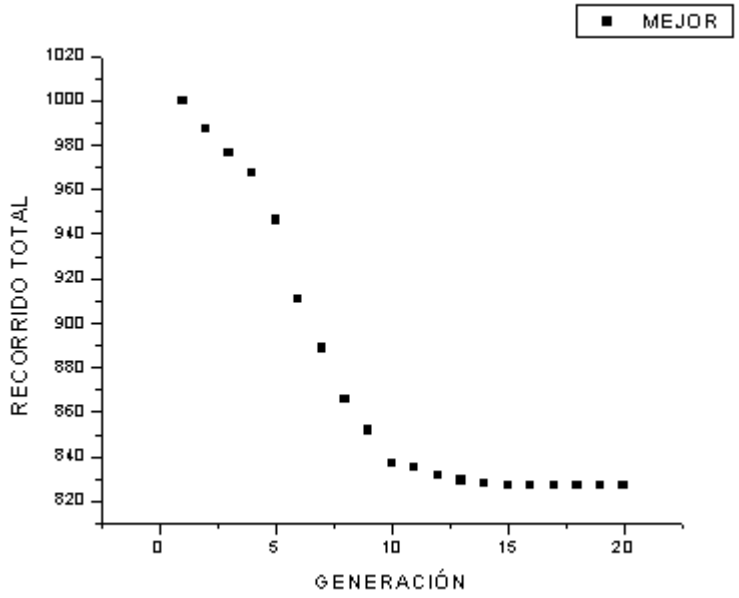

Figura 5.3. Resultados de GA-VRPTW por generación para instancia C105

Para la instancia C105 el mejor valor encontrado se presento a partir de la generación 15 a partir de ahí los valores se repitieron.

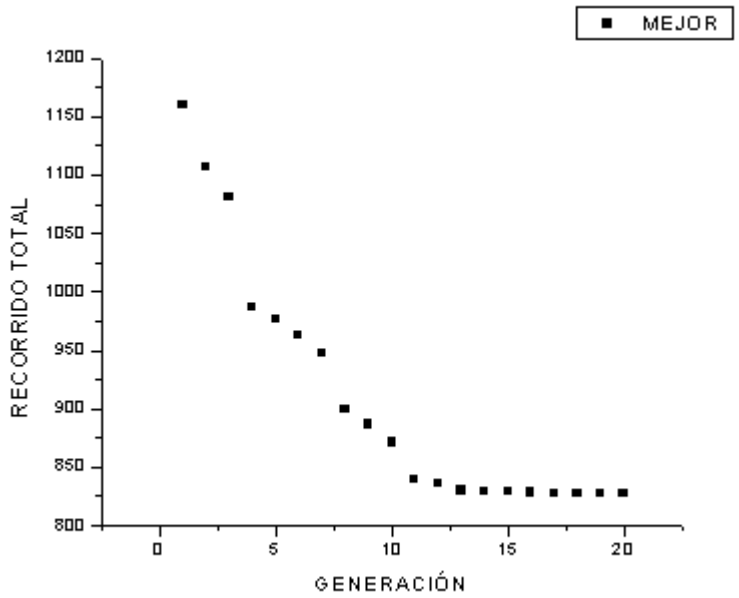

Figura 5.4. Resultados de GA-VRPTW por generación para instancia C108

Para la instancia C108 los valores convergieron partir de la generación 13 y de esa generación a la generación 20 los valores se repitieron.

Para la versión 3 (GA-VRPTW-SN) los datos ejecutados en el laboratorio de optimización y software se muestran en la Tabla 5.12 en comparación con otros algoritmos genéticos como: GenSAT [Thangiah, 1999], GIDEON [Thangiah 1993] y GENEROUS [Potvin y Bengio 1994].

GenSAT fue escrito en el lenguaje-c y en una computadora NeXT 68040 (25Mhz). GenSAT es una hibridación de varias técnicas como es genético, recocido simulado y búsqueda tabú. Los parámetros utilizados en el algoritmo genético para el tamaño de la población son de 1000 individuos, 50 generaciones, 0.6 de cruzamiento y 0.001 de mutación.

GIDEON fue implementado en un SOLBOURNE 5/802. GIDEON es una combinación de un genético con búsqueda local. Con 50 generaciones, un tamaño de población de 1000, un cruzamiento de 0.8 y una mutación de 0.001.

GENEROUS fue implementado en un Silicon Graphics Workstation. GENEROUS es una combinación de un genético con diferentes operadores de mutación y cruzamiento. Los valores tomados para la población son 150 individuos, con 50 generaciones, un cruzamiento de 0.6 y una mutación de 0.6.

En la Tabla 5.12 se observa que el algoritmo GA-VRPTW-SN generó buenos resultados en comparación con el algoritmo GENEROUS con lo que podemos decir que el algoritmo GA-VRPTW-SN es competitivo. Para los algoritmos GenSAT y GIDEON en eficiencia son superados por GA-VRPTW-SN y GENEROUS.

Tabla 5.12. Resultados de GA-VRPTW-SN en el laboratorio de optimización y software contra otros algoritmos genéticos

Problema	**GA-VRPTW-SN**		**GenSAT**		**GIDEON**		**GENEROUS**		**Mejor Conocido**	
	V	Mejor	V	Mejor	V	Mejor	V	Mejor	V*	Op*
C101	10	827.321	10	829	10	833	10	**827.3**	10	827.3
C102	10	827.378	10	829	10	832	10	**827.3**	10	827.3
C103	10	826.343	10	835	10	873	-	-	10	826.3
C104	10	823.090	10	835	10	904	-	-	10	822.9
C105	10	827.318	10	829	10	874	-	-	10	827.3
C106	10	**827.301**	10	829	10	902	10	**827.3**	10	**827.3**
C107	10		10	829	10	926	10	**827.3**	10	827.3
C108	10	827.338	10	829	10	928	10	**827.3**	10	827.3
C109	10		10	829	10	957	-	-	10	827.3

La convergencia de resultados por generación de algunas instancias se muestra en las Figuras 5.5 a 5.11. En estas figuras se muestra que con la implementación de la búsqueda local en vecindades a un genético la convergencia del mejor valor surge en las primeras generaciones en promedio en la quinta generación alcanza el mejor valor. En el caso de la versión GA-VRPTW la convergencia se realiza en promedio en la decima generación con este análisis se ve claramente que el algoritmo genético mejoró en eficiencia la búsqueda del mejor.

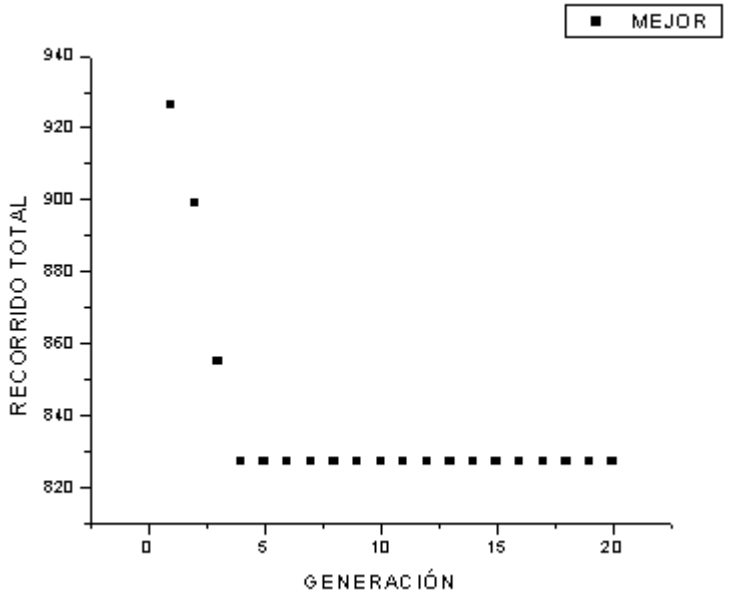

Figura 5.5. Resultados de GA-VRPTW-SN por generación instancia C101

La convergencia para la instancia C101 es en la generación 4.

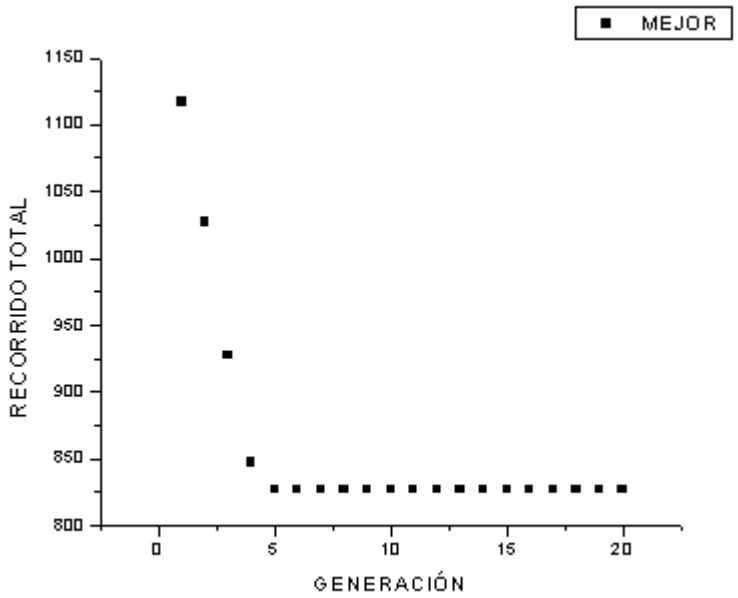

Figura 5.6. Resultados de GA-VRPTW-SN por generación para instancia C102-100

La convergencia para la instancia C102 es en la generación 5

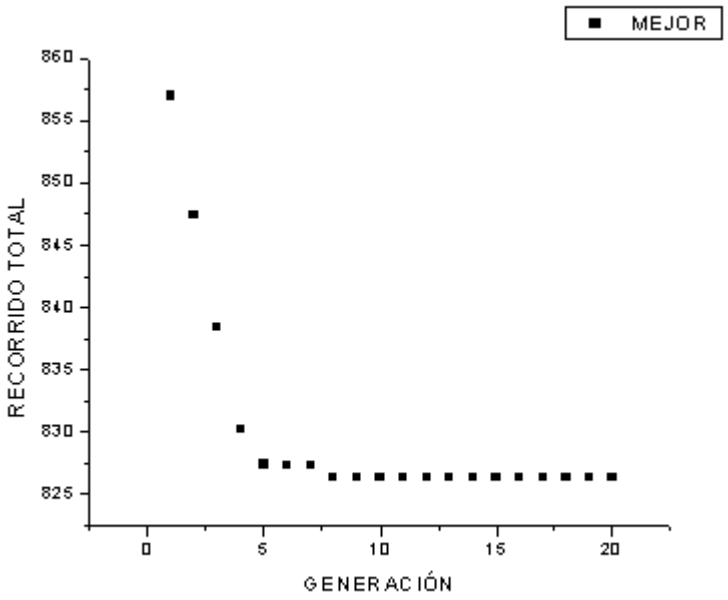

Figura 5.7. Resultados de GA-VRPTW-SN por generación para instancia C103-100

La convergencia de la instancia C103 es en la generación 5.

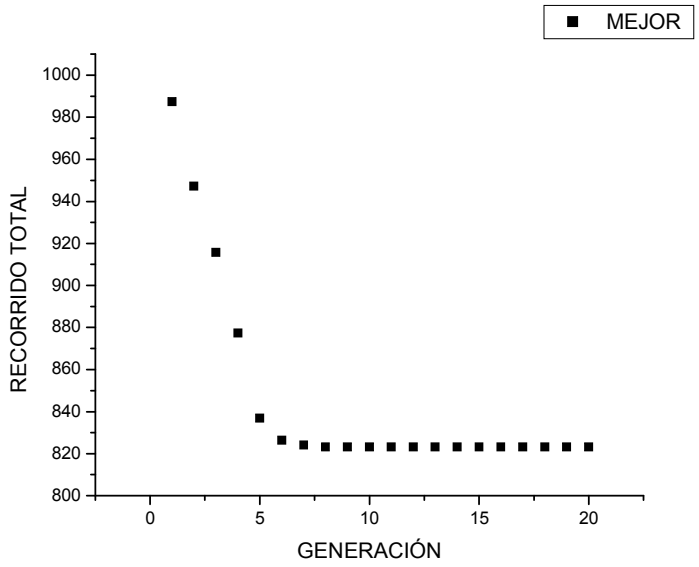

Figura 5.8. Resultados de GA-VRPTW-SN por generación para instancia C104-100

La convergencia para la instancia C104 es en la generación 7.

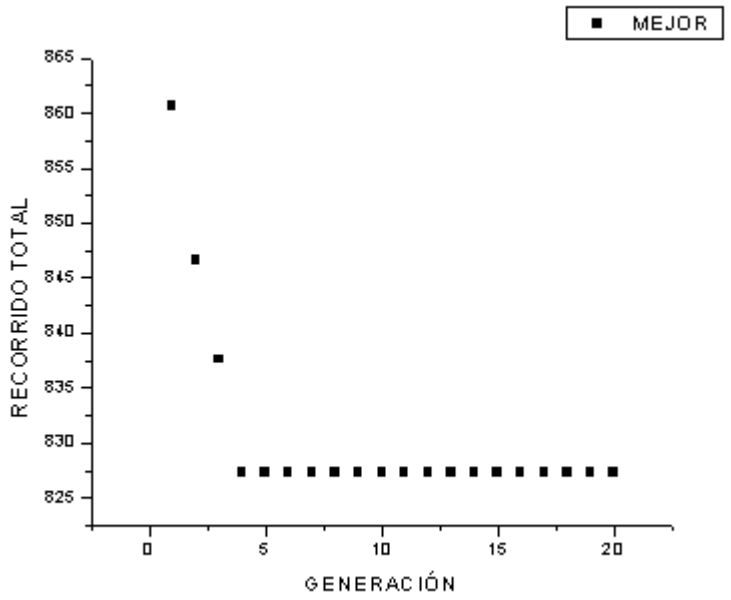

Figura 5.9. Resultados de GA-VRPTW-SN por generación para C105-100

La convergencia de la C105 es en la generación 4.

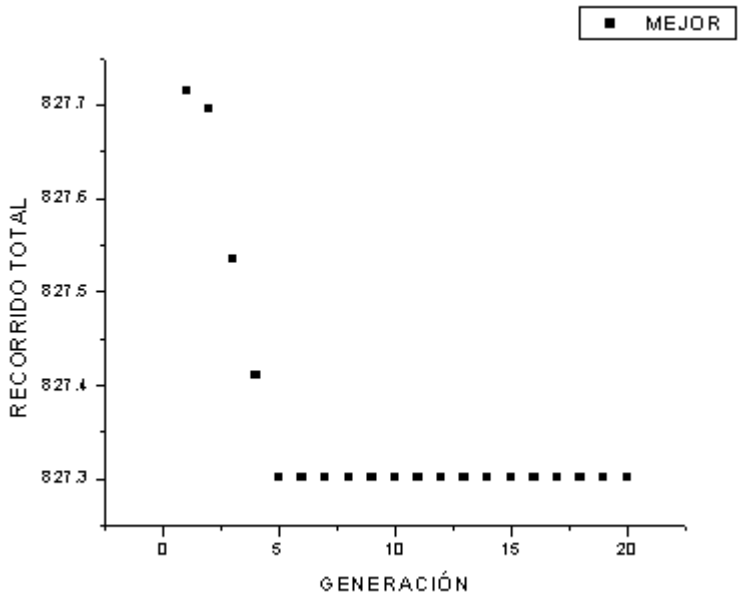

Figura 5.10. Resultados de GA-VRPTW-SN por generación para C106-100

La convergencia de C106 es en la generación 5.

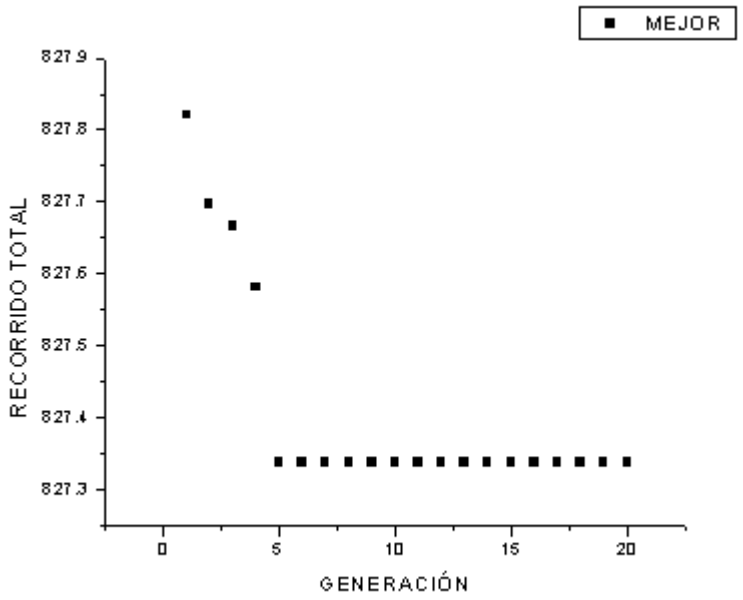

Figura 5.11. Resultados de GA-VRPTW-SN por generación para C108

La convergencia de C108 es en la generación 5.

En cuanto al tiempo de ejecución del algoritmo GA-VRPTW contra el algoritmo GA-VRPTW-SN se muestran en la Figura 5.12. Donde se observa que con la estrategia de vecindad se mejora el algoritmo genético por que toma menor tiempo en converger al mejor reportado. Se graficaron los tiempos por generación de la instancia C101 obtenidos por el algoritmo GA/VRPTW y los tiempos por generación de la instancia C106.obtenidos por el algoritmo GA-VRPTW-SN.

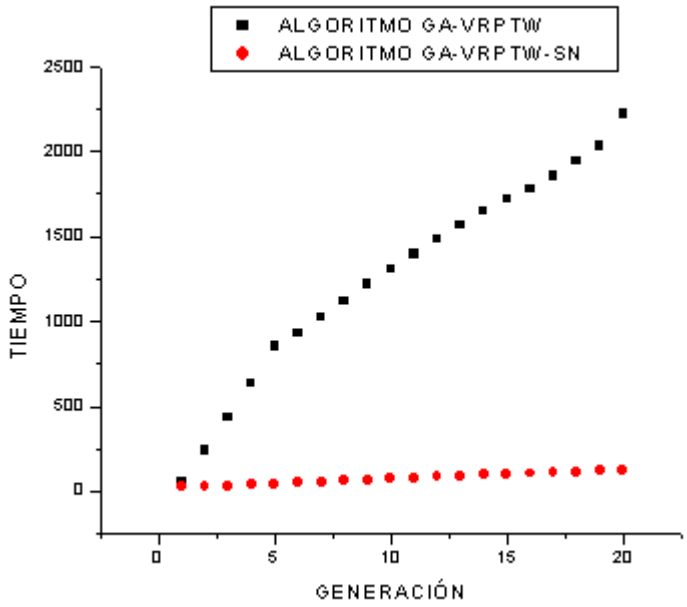

Figura 5.12. Resultados de tiempos de ejecución por generación de los algoritmos GA-VRPTW contra GA-VRPTW-SN

Los resultados obtenidos del algoritmo GA-VRPTW-SN en la computadora Aleph se presentan en la Tabla 5.13 en comparación con los obtenidos en el laboratorio de optimización y software. Se observa que el algoritmo ejecutado en la computadora Aleph mejoró en tiempo los resultados del mismo algoritmo ejecutado en el laboratorio de optimización y software. El valor obtenido por el algoritmo ejecutado en la computadora Aleph para el recorrido total fue mejor que el obtenido por el algoritmo ejecutado en el laboratorio de optimización y software.

Tabla 5.13. Resultados de GA-VRPTW-SN en la computadora Aleph contra resultados del laboratorio de Optimización y Software

Instancia	**GA-VRPTW-SN Aleph**		**GA-VRPTW-SN Laboratorio de optimización y software.**		Op*
	Mejor	t, sec	Mejor	t, sec	
C101-100	827.338	382.75	827.321	784.07	827.3
C102-100	**827.307**	410.75	827.378	890.09	827.3
C103-100	826.320	448.5	826.343	1089.71	826.3
C104-100	822.992	503.75	823.090	1033.20	822.9
C105-100	**827.305**	563.5	827.318	1213.37	827.3
C106-100	**827.307**	441.5	827.301	941.37	827.3
C107-100					827.3
C108-100	827.313	646.75	827.338	5883.23	827.3
C109-100					827.3

Para visualizar mejor la ganancia en tiempo del algoritmo GA-VRPTW-SN contra el algoritmo GA-VRPTW, se construyó la Figura 5.13

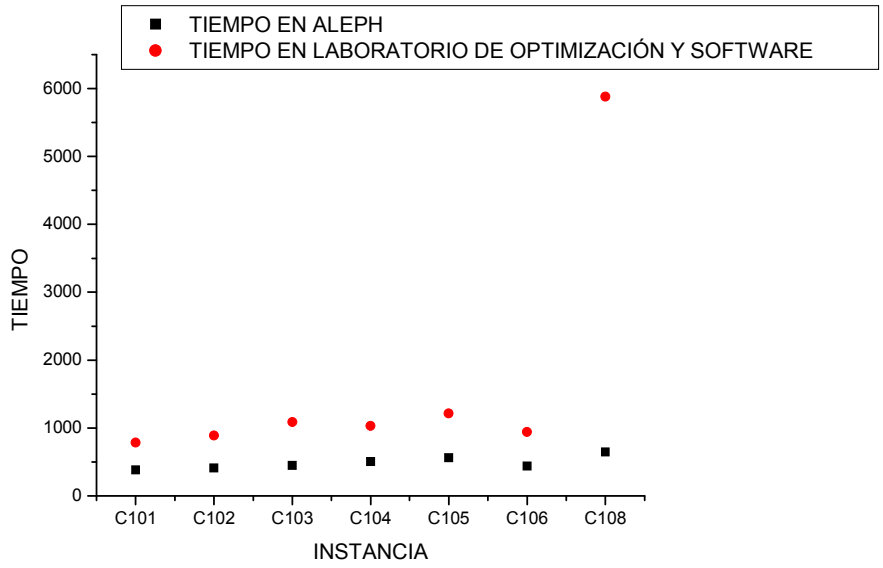

Figura 5.13. Tiempos de ejecución del algoritmo GA-VRPTW-SN en Aleph y Laboratorio de Optimización y software

En la Figura 5.13 se muestra la gráfica que representa el comportamiento del algoritmo GA-VRPTW-SN en diferentes equipos (Aleph y laboratorio de optimización y software).

Para la versión GA-VRTW-PN en paralelo el análisis de los resultados se muestra en la siguiente sección.

5.4.2. Análisis comparativo con algoritmos en paralelo

En esta sección se presenta un análisis de los resultados del algoritmo propuesto por el autor como una solución al problema del transporte con ventanas de tiempo. Los resultados del algoritmo GA-VRPTW-PN se muestran en la Tabla 5.14 en comparación de tiempos con diferente número de procesadores. Algunas de las pruebas se realizaron con 4, 8 y 32

procesadores. Los parámetros de entrada de estas pruebas fueron con una población de 1000 individuos, 20 generaciones, 0.01 de cruzamiento y un tamaño de muestra en vecindad de 10 individuos.

Tabla 5.14. Resultados de GA-VRPTW-PN en computadora Aleph con diferente número de procesadores.

Benchmark	**GA-VRPTW-VP** **Con 4 procesadores**		**GA-VRPTW-VP** **Con 8 procesadores**		**GA-VRPTW-VP** Con 32 procesadores		Op
	Mejor	t, sec	Mejor	t, sec	Mejor	t, sec	
C101-100	**827.327**	1090.5	827.300	690.2	827.301	144	827.3
C102-100	827.302	1002.57	827.301	924.3	827.300	620	827.3
C103-100	826.300	1064.2	826.300	5750.5	826.303	413	826.3
C104-100			826.000	5216.6	822.905	309	822.9
C105-100	**827.300**	1311.5	827.303	1501.03	827.341	361	827.3
C106-100	827.305	1082.6	827.302	1241.5	827.305	267	827.3
C107-100	827.301	3509.2	827.303	9125.65	827.300	1484	827.3
C108-100	**827.315**	8591.4	827.301	4288.08		1665	827.3
C109-100							

El tiempo de ejecución del algoritmo genético GA-VRPTW-PN para cada una de las instancias reportadas en la Tabla 5.14 utilizando diferente número de procesadores deja ver que a mayor número de procesadores menor es el tiempo de ejecución como se muestra en la Figura 5.14.

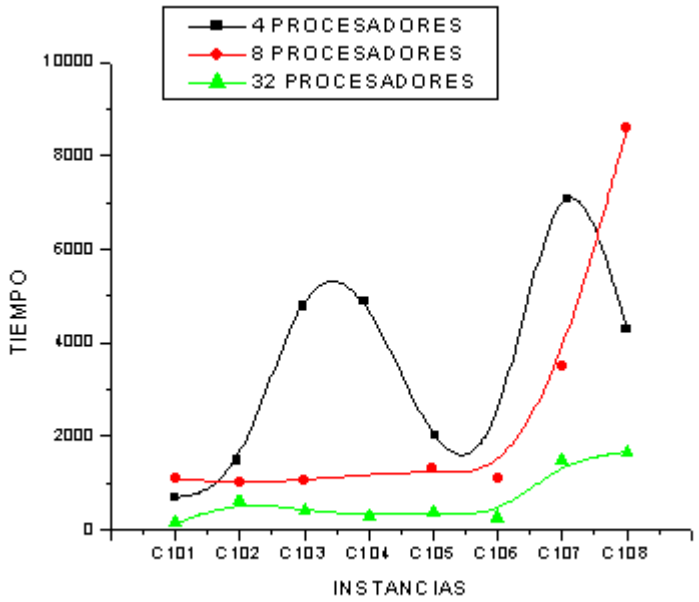

Figura 5.14. Tiempo de GA-VRPTW-PN con diferente número de procesadores

Los tiempos de ejecución con 32 procesadores son menores ya que la carga de trabajo para cada procesador es menor ya que se encuentra distribuida entre el total de procesadores.

Las comparaciones de los resultados del algoritmo GA-VRPTW-PN ejecutados en la computadora Aleph se presentan en la Tabla 5.15. La paralelizacion se llevo a cabo con motivos de hacer más eficiente el algoritmo GA-VRPTW-SN comparado con otros algoritmos genéticos en paralelo reportados en literatura. El algoritmo con el que se comparo es el algoritmo BBB [Berger et al., 2004] ejecutado en un equipo Pentium 400 MHz (54 Mflops/s).

Los resultados obtenidos por el algoritmo GA-VRPTW-PN son mejores que los obtenidos por el algoritmo BBB, cabe mencionar que el algoritmo BBB utiliza también un algoritmo genético.

Tabla 5.15. Comparación de resultados del algoritmo GA-VRPTW-PN contra algoritmos genéticos reportados en literatura

Instancia	GA-VRPTW-VP		BBB		Op*
	Mejor	t, sec	Mejor	t, sec	
C101-100	827.301	144	828.50	1800	827.3
C102-100	827.300	620	828.50	1800	827.3
C103-100	826.303	413	828.50	1800	826.3
C104-100	822.905	309	828.50	1800	822.9
C105-100	827.341	361	828.50	1800	827.3
C106-100	827.305	267	828.50	1800	827.3
C107-100	827.300	1484	828.50	1800	827.3
C108-100		1665	828.50	1800	827.3
C109-100	827.301	144	828.50	1800	827.3

Con motivo de analizar el comportamiento de la versión GA-VRPTW-SN contra la versión GA-VRPTW-PN se construye la Tabla 5.16. Para observar el comportamiento de las versiones y ver cuál de las dos cumple con los objetivos planteados inicialmente. El algoritmo GA-VRPTW-PN paralelo tiene mejores resultados que la versión secuencial GA-VRPTW-SN y supera en eficiencia y eficacia a otros algoritmos reportados en literatura. Entonces podemos decir que el algoritmo GA-VRPTW-VP resulto eficaz y eficiente con lo que se cumple el objetivo planteado.

Por toda la experimentación realizada cabe mencionar que la realización de las diferentes versiones fue para finalmente conjuntar lo bueno da cada una en una sola versión que en este caso resultó ser el algoritmo GA-VRPTW-PN que cumple con el objetivo planteado inicialmente.

Tabla 5.16. Comparación de resultados de algoritmo GA-VRPTW-VS contra algoritmo GA-VRPTW-VP ejecutados en la computadora Aleph

Instancia	GA-VRPTW-VS		GA-VRPTW-VP		Op*
	Mejor	Tiempo en segundos	Mejor	Tiempo en segundos	
C101-100	**827.338**	382.75	827.301	144	827.3
C102-100	827.307	410.75	827.300	620	827.3
C103-100	826.320	448.5	826.303	413	826.3
C104-100	822.992	503.75	822.905	309	822.9
C105-100	**827.305**	563.5	827.341	361	827.3
C106-100	827.307	441.5	827.305	267	827.3
C107-100			827.300	1484	827.3
C108-100	**827.313**	646.75		1665	827.3
C109-100			827.301	144	827.3

El algoritmo GA-VRPTW-PN muestra mejores resultados de eficiencia y eficacia que la versión secuencial. Es más eficiente debido a la distribución de cargas por procesador y por la velocidad de procesamiento.

En la Figura 5.15 se muestra la eficacia de los algoritmos GA-VRPTW-SN y GA-VRPTW-PN. En cuanto a eficacia se aprecia que ambos algoritmos dan buenos resultados y que la variación entre ellos es mínima.

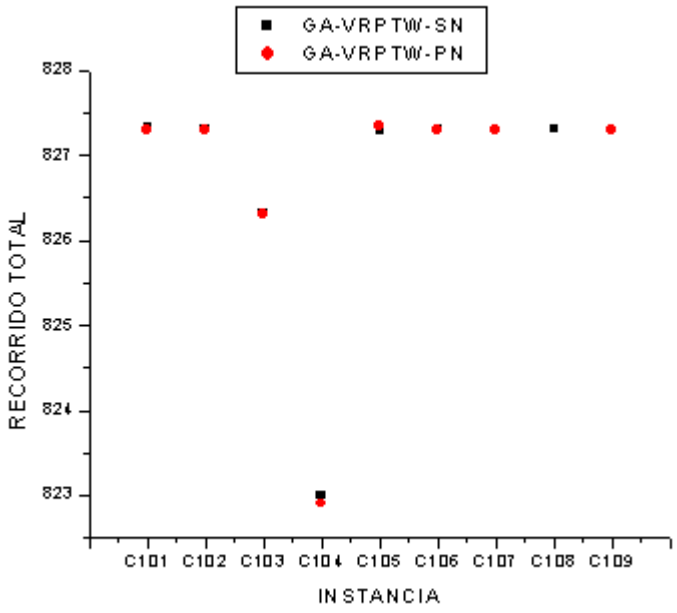

Figura 5.15. Resultados de eficacia de GA-VRPTW-SN contra GA-VRPTW-PN en computadora Aleph

En cuanto a eficiencia se observa en la Figura 5.16 que el algoritmo en paralelo es más rápido que el algoritmo secuencial, ambos algoritmos ejecutados en las mismas condiciones. Por lo que se puede decir que el algoritmo GA-VRPTW-PN resulta ser más eficiente que el algoritmo GA-VRPTW-SN.

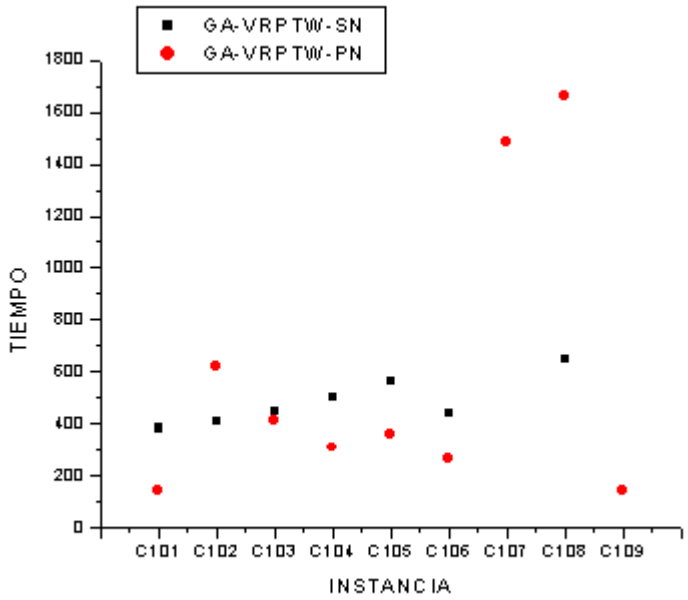

Figura 5.16. Resultados de eficiencia de GA-VRPTW-SN contra GA-VRPTW-PN en computadora Aleph

Capítulo 6. Conclusiones

6.1. Contribuciones

Proponer una alternativa de solución a problemas NP no es una tarea fácil de concluir. En este trabajo de investigación para poder llegar a la alternativa de solución propuesta se realizaron varias combinaciones de estrategias de solución mediante heurísticas. En este recorrido de búsqueda de soluciones se presentan las siguientes contribuciones:

1) Una alternativa de solución al problema del transporte con ventanas de tiempo mediante un algoritmo evolutivo secuencial.
2) Una alternativa de solución al problema del transporte con ventanas de tiempo mediante un algoritmo evolutivo paralelo.
3) Un operador cruzamiento-k sometido a restricciones propias del problema del transporte con ventanas de tiempo.
4) Un operador mutación-S con comportamiento inteligente.
5) La hibridación de un algoritmo evolutivo con técnicas de búsqueda local en vecindades.
6) La hibridación de un algoritmo evolutivo con técnica de satisfacción de restricciones para generar la población inicial.
7) La hibridación de un algoritmo evolutivo con técnica de minería de datos para generar la población inicial.
8) Una vecindad modificada tipo OR para el problema del transporte con ventanas de tiempo.
9) Una metodología para convertir algoritmos secuenciales en algoritmos paralelos.
10) Análisis de problemas de optimización combinatoria mediante metodologías Checklan para la obtención de variables principales.

Todas las contribuciones presentadas aportaron información relevante a la investigación y fueron pieza clave para el cumplimiento del objetivo originalmente planteado.

6.2. Recomendaciones

Por la investigación y experimentación realizada en este documento se recomienda la utilización de metodologías para el análisis del problema. Una vez analizado teóricamente el problema y entendido su comportamiento; el siguiente paso en construir el algoritmo y su diagrama de flujo de información requerida, después de entender el flujo de información se procede a seleccionar la herramienta de desarrollo (según preferencias del programador) y la siguiente fase es la realización de pruebas y comparación de resultados. Tomando en cuenta esta secuencia de actividades es posible concluir satisfactoriamente el proceso de búsqueda de solución a cierto problema planteado.

Otra parte medular en el desarrollo de una investigación es el estado del arte que permite detectar el nicho de investigación a enfocarse y la creación de nuevas ideas como alternativas de solución y aportaciones a la comunidad científica.

En las propuestas de solución es importante tomar en cuenta el grado de complejidad del problema y las herramientas y técnicas para la construcción de una posible solución. Tener presente también que la solución a problemas combinatorios mediante heurísticas es aceptable siempre y cuando se utilice la combinación correcta de técnicas dependiendo del problema que se esté abordando. Muchas veces la combinación correcta puede ser guiada, es decir estudiar las propuestas de otros autores y

proponer en base a esas investigaciones una continuidad, o puede ser totalmente a prueba y error, es decir creando cosas nuevas y validándolas con los trabajos propuestos por otros autores.

Como recomendación final se sugiere utilizar programas de actividades para la organización del trabajo, para el desarrollo, para la implementación, para el análisis de resultados, para la escritura del reporte y en conclusión para toda actividad que circunde nuestro espacio existencial.

6.3. Trabajos futuros

La solución propuesta en este trabajo de investigación sirve como antecedente para las siguientes actividades:

1) Adecuar el algoritmo genético paralelo para aplicarlo a otros problemas NP con el objetivo de generar otras alternativas de solución a este tipo de problemas.
2) Proponer un criterio de paro en vecindades.
3) Proponer estrategia de selección de muestras en vecindades.
4) Aplicar el algoritmo propuesto a instancias reales de logística y distribución en las empresas.
5) Definir instancias reales para utilizarlas en el algoritmo propuesto.
6) Aplicar el algoritmo para construcción de rutas de transporte escolar.
7) Expreimentación par ainstancias Solomon-extendidas propuestas por Homberguer [Homberger, Gehring. 1999] en donde se propone el aumento de número de clientes. Con estas instancias de 200, 400, 600, 800 y 1000 clientes aun se siguen trabajando, hasta la fecha aun no se reportan resultados de mejores encontrados en literatura.

6.4. Conclusiones

En respuesta a la hipótesis planteada. Sí es posible construir una solución combinando métodos, técnicas y herramientas dentro del ámbito computacional que den una solución acertada al problema del transporte.

Del trabajo realizado se deriva que la solución propuesta para el problema de transporte con ventanas de tiempo en su versión paralela (GA-VRPTW-PN) es eficaz, eficiente y competitivo con los mejores resultados de los hasta ahora conocidos para el VRPTW. Los resultados fueron comparados con soluciones propuestas por otros autores que han trabajado con heurísticas para el problema de transporte con ventanas de tiempo.

La solución propuesta al problema del transporte con ventanas de tiempo (GA-VRPTW-PN) es una heurística compuesta de un algoritmo genético con un algoritmo de búsqueda en vecindades e implementado de forma paralela utilizando la técnica OpenMP. El algoritmo genético tiene la siguiente combinación de operadores:

- El operador de selección utiliza la técnica de torneo.
- El operador de cruzamiento utiliza la técnica cruzamiento-k.
- El operador de mutación la técnica mutación-S.

El algoritmo de búsqueda en vecindades utiliza una vecindad modificada tipo OR adecuada al problema del transporte con movimiento 1-óptimos.

La técnica openMP tiene la particularidad de que los procesos acceden a una memoria común por lo que se tiene que cuidar el proceso de sincronización.

Otra de las cosas que podemos decir con respecto al trabajo realizado es que el proceso de paralelización en optimización combinatoria permite

aumentar la eficiencia del algoritmo, siempre y cuando el diseño del algoritmo desde inicio tenga una complejidad polinómica o logarítmica, lo ideal sería que fuese de grado logarítmico para garantizar un mejor desempeño del algoritmo.

El hecho de que se paralelice un algoritmo no quiere decir que se disminuye el grado de complejidad del algoritmo el grado de complejidad del algoritmo sigue siendo el mismo lo que cambia es la ecuación de tiempo de ejecución o función temporal ya que en paralelo las tareas se distribuyen en varios procesadores.

Los algoritmos genéticos por su naturaleza permiten trabajar los datos de manera independiente en cada una de sus fases, lo que los hace altamente paralelizables.

El problema del transporte con ventanas de tiempo es un problema difícil de resolver por las restricciones y variables que lo describen. Sin embargo haciendo uso de técnicas heurísticas combinadas de las existentes en literatura o proponiendo nuevas técnicas, se logran acercamientos a los mejores resultados reportados. El algoritmo propuesto (GA-VRPTW-PN) en este trabajo de investigación resuelve instancias propuestas por Solomon de las clasificadas como grandes de 100 nodos.

REFERENCIAS

[Affenzeller, 2002] M. Affenzeller. (2002). “A Generic Evolutionary Computation Approach Based Upon Genetic Algorithms and Evolution Strategies”. Journal of Systems Science, vol. 28, no. 2, pp. 59-72. 2002.

[Alba et al., 2003] Enrique Alba Francisco Luna Antonio J. Nebro. (2003).Parallel Heterogeneous Genetic Algorithms for Continuous Optimization. Departamento de Lenguajes y Ciencias de la Computaci´on E.T.S. Ingenier´ıa Inform´atica. Campus de Teatinos, 29071 M´alaga (Espa˜na). feat,flv,antoniog@lcc.uma.es. ScienceDirect Parallel Computing. Volume 30, Issues 5-6, May-June 2004, Pages 699-719. Parallel and nature-inspired computational paradigms and applications

[Alegre et al., 2002] J. F. Alegre, S. Casado, C. R. Delgado (2002). Diseño de calendarios para transporte de componentes de automóviles. Soluciones heurísticas. Volumen 20 II pp. 301-316

[Antún y Briceño, 2001] J. Antún, S. Briceño. (2001). Énfasis Logística México | Año II | Número 7 Enero 2001 | DistribuciónTercerización: nueva visión de la cadena de suministro.

[Arbelaitz et al., 2001] O. Arbelaitz, C. Rodriguez, I. Zamkola. (2001). Low cost parallel solutions for the VRPTW optimization problem. Fourth International Conference on Parallel Processing Workshops, 2001, pp.176-181.

[Archetti et al., 2001] C.Archetti, R. Mansini, M. G. Speranza. (2001). The Split Delivery Vehicle Routing Problem with Small Capacity, November 7, 2001. Disponible en: http://fausto.eco.unibs.it/~www_mequ/ricerca/quaderni/201.pdf. Consultado en: Febrero de 2005.

[Balakrishnan, 1993] N. Balakrishnan. (1993). Simple heuristics for the vehicle routing problem with soft time windows. Journal of the Operational Research Society, 44(3):279 { 287, 1993.

[Balbuena y Herrera, 2003] A. Balbuena, A. Herrera. (2003). Manual Estadístico del Sector de Transporte, colaboradores Maria del Carmen E.,

Guadalupe Morales, Agustín Bustos, Carlos Martner, Aurora Moreno, Instituto Mexicano del Transporte (IMT), México, DF.

[Balseiro, 2006] S. Balseiro. (2006). Algoritmos para problemas de ruteo de vehículos con restricciones de capacidad y ventanas de tiempo. Argentina Chile Uruguay

[Balseiro, 2007] S. Balseiro. (2007). Un algoritmo de colonia de hormigas para el problema de ruteo con costos dependientes del tiempo y con ventanas de tiempo (TDVRPTW), 2007. [Orientadores: I. Loiseau/Juan Ramonet, UBA]

[Baran y Schaerer, 2003] B. Baran, M. A. Schaerer.(2003). Multiobjective Ant Colony System for Vehicle Routing Problem with Time Windows. Proceedings of the 21st IASTED International Conference APPLIED INFORMATICS, Innsbruck, Austria.

[Barreto-Sedeño, 2008] E. Barreto-Sedeño. (2008). Análisis experimental de estructuras de vecindad en algoritmos evolutivos. UAEM-CIICAp Diciembre 2008.

[Bastos et al., 2005] Guilherme Bastos Alvarenga, Ricardo Martins de Abreu Silva, Rudini Menezes Sampaio.(2005). A Hybrid Algorithm for the Vehicle Routing Problem with Time Window. Dept. of Computer Science, Federal University of Lavras, Brazil {guilherme, rmas, rudini} @ dcc.ufla.br

[Berger et al., 2004] Jean Berger, Mohamed Barkaoui, Olli Braysy. (2004).A Parallel Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows. Elsevier. Computers & Operations Research. Volume 31, Issue 12, October 2004, Pages 2037-2053

[Bianchi, 2005] L. Bianchi. (sa). Metaheuristics for the Vehicle Routing Problem with Stochastic Demands; S/F., Disponible en: http://www.idsia.ch/idsiareport/IDSIA-06-04.pdf., Consultado en: Febrero de 2005.

[Brando et al., 2006] Humberto Cesar Brando de Oliveira, Germano Crispim Vasconcelos, Guilherme Bastos Alvarenga, "A Multi-Start Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows," sbrn,pp.137-142, Ninth Brazilian Symposium on Neural Networks (SBRN'06), 2006

[Bräysy y Gendreau, 2001] O. Bräysy, M. Gendreau. (2001). Metaheuristics for the vehicle routing problem with time windows, TechnicaL Report STF42 A01025, SINTEF Applied Mathematics, Department of Optimization, Norway. 2001.

[Bräysy y Gendreau, 2001 b] O. Bräysy, M. Gendreau. (2001). Route construction and local search algorithms for the vehicle routing problem with time windows, Report STF42 A01024, App. Mathematics, Dep. of Optimization, Norway. 2001.

[Bräysy y Gendreau, 2002] O. Bräysy, M. Gendreau. (2002). Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows. Localización: Top, ISSN 1134-5764, Vol. 10, Nº. 2, 2002 , pags. 211-237

[Bräysy, 2001] O. Bräysy. (2001). Genetic Algorithms for the Vehicle Routing Problem with Time Windows. 2001, Disponible en: http://osiris.tuwien.ac.at/~wgarn/VehicleRouting/Braysy.pdf, Consultado en: Febrero de 2005.

[Cantu-Paz, 1998] E. Cantu-Paz. (1998). A summary of parallel genetic algorithms. Calculateurs Parallels Reseaux et Systemes Repartis, 10:141-170.

[Chafekar et al., 2003] D. Chafekar, J. Xuan, K. Rasheed. (2003). “Constrained Multi-objective Optimization Using Steady State Genetic Algorithms”, Computer Science Departament University of Georgia. Athens, Genetic and Evolutionary Comput ation Confer en ce, GA 30602 USA.

[Charlotte y Goetschalckx, 1998] J. Charlotte, M. Goetschalckx. (1998). The vehicle routing problem with backhauls: properties and solution algorithms. Scholl of industrial and systems engineering - georgia institute of technology, Copyright 1992 – 1998. Disponible en: http://www.isye.gatech.edu/people/faculty/Marc_Goetschalckx/cali/Lineback/Vehicle%20Routing%20Problem%20with%20Backhauls.pdf, Consultado en Febrero de 2005.

[Chin et al., 1999] A. Chin, H. Kit, A. Lim.(1999). A new GA approach for the vehicle routing problem. 11th IEEE International Conference on Tools with Artificial Intelligence, 1999, pp.307-310.

[Chong, 2004] W. W. Chong (2004). Meta-heuristics development framework: Design and aplications. A thesis submited for the degree of master of science school of computing national university of singapore.

[Coello, 2008] C. A. Coello Coello. (2008) "Introducción a la Computación Evolutiva. Notas de Curso", Departamento de Computación,CINVESTAV-IPN, México, D.F.

[Conacyt, 2008] Fondo Sectorial de Investigación S.R.E.-CONACYT http://132.248.31.69/index.php?option=com_content&task=view&id=224&Itemid=1. http://www.conacyt.mx/Fondos/Sectoriales/SRE/2008-01/SRE_2008-1_Bases-Convocatoria.pdf

[Conacyt, 2008 b] Departamento Ingeniería de Electrónica e Informática (DEI), Ing. Vicente González, tel. 334650 int. 101, correo-e:vgonzale@uca.edu.py http://www.uca.edu.py/cyt/index.php?id=84

[Conway, 1963] M. E. Conway. (1963). A multiprocessor system design. In proceedings of AFIPS fall joint computer conference, pp 139-146.

[Cook, 1971] S.A. Cook. (1971) The complexity of theorem proving procedures Proc. 3rd ACM symposium on theory of computing. Shaker Heights, Ohio. 151-158

[Cortéz, 2004] A. Cortéz. (2004). Teoría de la complejidad computacional y teoría de la computabilidad, Revista investigación y sistemas de información pp. 102-105 ISSN 1815-0268

[Crainic y Toulouse, 2003] T. G. Crainic, M. Toulouse. (2003). Parallel strategies for metaheristics, volumeHandbook of metaheuristics, pp 475-513. Kluwer academic publisher.

[Cruz y Moreno, 2007] C. Cruz Corona, J. M. Moreno. (2007). Estrategias cooperativas paralelas en la solución de problemas de optimización, Revista Iberoamericana de Inteligencia Artificial, ISSN 1137-3601, Nº. 34, Granada, España, pp. 129-143.

[Cruz-Chávez y Díaz-Parra, 2008] M. A. Cruz-Chávez, O. Díaz-Parra. (2008). “Evolutionary Algorithm for the Vehicles Routing Problem with Time Windows Based on a Constraint Satisfaction Technique”. Journal computation and systems. Appear. 2008.

[Cruz-Chávez et al., 2008] Marco Antonio Cruz-Chávez, Ocotlán Díaz-Parra, David Juárez Romero,Eric Barreto Sedeño, Crispín Zavala Díaz, Martín G. Martínez Rangel, Un Mecanismo de Vecindad con Busqueda Local y Algoritmo Genético para Problemas de Transporte con Ventanas de Tiempo, CICos 2008, 6to Congreso Internacional de Cómputo en Optimización y Software, ACD, ISBN(i) , ISBN(e), pp., 25-27 Junio, México, 2008 (Proceedings)

[Cruz-Chávez et al., 2008 b] Marco Antonio Cruz-Chávez, Ocotlán Díaz-Parra, David Juárez-Romero, Martín G. Martínez-Rangel: Memetic Algorithm Based on a Constraint Satisfaction Technique for VRPTW. Serie: Lecture Notes in Artificial Intelligence. Book: Artificial Intelligence and Soft Computing – ICAISC 2008. Springer Verlag Pub. Berlin Heidelberg. ISSN: 0302-9743. Vol.5097, pp. 376-387, 2008. (chapter of book)

[Cruz-Chávez et al., 2007] Marco Antonio Cruz-Chávez, Ocotlan Díaz-Parra, J.A.Hernández, José Crispin Zavala-Díaz, Martín G Martínez-Rangel: Search Algorithm for the Constraint Satisfaction Problem of VRPTW. Electonics, Robotics and Automative Mechanics Conference (CERMA 2007). IEEE-Computer Society, ISBN: 0-7695-2974-7, pp 746-751, 25-28 September, México, 2007. Indexs: ISTP/ISI Proceedings. INSPEC. (ISI Proceedings)

[Czech y Czarnas, 2002] Z.J. Czech, P. Czarnas. (2002). Parallel simulated annealing for the vehicle routing problem with time windows. 10th Euromicro Workshop on Parallel, Distributed and Network-based Proceedings, 2002, pp.376-383.

[Dantzing y Ramser, 1959] G. B. Dantzing, J. H. Ramser. (1959). "The Truck Dispatching Problem". Management Science 6 (1): 80-91. Retrieved on 2008-04-17.

[DeJong, 1975] A. K. De Jong. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems.PhD thesis, University of Michigan.

[Delgado y Pacheco, 2001] C. Delgado, J. Pacheco. (2001). “Minmax vehicle routing problems: application to school transport in the province of Burgos (Spain)”, Lecture Notes in Economics and Mathematical Systems, 505, pp. 297-318.

[Dethloff, 2002] J. Dethloff. (2002). Relation between vehicle routing problems: an insertion heuristic for the vehicle routing problem with

simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls. En: Journal of the Operational Research Society, 2002.

[Díaz et al., 1996] A. Díaz, F. GLover, M. Hassan, M. Ghaziri, J.L. Gonzalez Velarde. (1996). Procedimientos mas conocidos o reconocidos en optimizacion combinatoria Optimización Heurística y Redes Neuronales Edited by Adenso Díaz Authored by Adenso Díaz, Fred Glover, Hassan M. Ghaziri, José Luis González Velarde, Manuel Laguna, Pablo Moscato , and Fan T. Tsen Published July 1996.

[Díaz-Parra y Cruz-Chávez, 2008] Ocotlán Díaz-Parra, Marco A. Cruz-Chávez: Evolutionary Algorithm with Intelligent Mutation Operator that solves the Vehicle Routing Problem of Clustered Classification with Time Windows. Polish Journal of Environmental Studies, Vol. No., Hard (2008) ISSN:1230-1485. (Journal ISI)

[Díaz-Parra y Cruz-Chávez, 2008 b] Ocotlán Díaz-Parra, Marco A. Cruz-Chávez: General Methodology to Parallelize with OpenMP Evolutionary Algorithms Applied to Combinatorial Optimization Problems. Polish Journal of Environmental Studies, Vol. No., Hard (2008) ISSN:1230-1485. (Journal ISI)

[Díaz-Parra et al., 2006] Ocotlán Díaz-Parra, Marco Antonio Cruz-Chávez, Damaris Galván-Montiel, José Crispín Zavala-Díaz: Técnicas de Modelación de Sistemas Blandos Aplicadas al Problema del Transporte Escolar, 5to Congreso de Cómputo AGECOMP, UAEM, ISBN(i)970-9750-02-X, ISBN(e) 968-878-273-4, pp. 49-62, 21-23 Noviembre, México, 2006. (Proceedings)

[Dorronsoro-Diaz, 2007] B. Dorronsoro-Diaz. (2007). Sitio web para VRP. This site was made in collaboration between AUREN and the Languages and Computation Sciences department of the University of Málaga by Bernabé Dorronsoro Díaz (Personal homepage)

[Flynn y Rudd, 1996] M.J. Flynn, K.W. Rudd. (1996). Parallel architectures. ACM computing surveys, 28(1): 67-70.

[Flynn, 1972] M. J. Flynn. (1972). Some Computer Organizations and their Effectivenes. IEEE transactions on computing, 21(9):948-948.

[Francis et al., 2004] P. Francis, K. Smilowitz, M. Tzur. (2004). The period vehicle routing problem with service choice; 2004., Disponible en: http://www.iems.northwestern.edu/ images/PDF/WP_04_005.pdf, Consultado en: Julio 2008.

[Gambardella et al., 1999 b] L. M. Gambardella, E. Taillard, G. Agazzi. (1999). MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows, New Ideas in Optimization, Disponible en: http://www.idsia.ch. McGraw-Hill, 1999.

[Gambardella et al., 1999] L. M. Gambardella, E. Taillard, G. Agazzi. (1999). News ideas in optimization. Mac Graw-Hill: London, 1999, pp.73-76.

[García, 2005] J. P. García Sabater. (2004-2005). Modelos y métodos de investigación de operaciones. Procedimientos para pensar. (modelado y resolución de problemas de organización industrial mediante programación matemática) Métodos cuantitativos de organización industrial

[Garey y Johnson, 1979] M. R. Garey, D. Johnson. (1979). ”Computers and intractability” a guide to the theory of NP-Completeness. Freeman, New York NY. ISBN 0-7167-1044-7

[Garey y Johnson, 2003] M. R. Garey, D. Johnson. (2003). "Computers and intractability, A Guide to the theory of NP-Completeness". Michael R. Garey / David S. Johnson. eds. W.H.FREEMAN AND COMPANY, New York M. R. Garey M.R., Jhonson D.S.

[Gen y Cheng, 2000] M. Gen, R. Cheng. (2000). Genetic Algorithms & Engineering Optimization.Wiley Series in Engineering Design and Automation. John Wiley & Sons, NewYork, 2000.

[Gendreau et al., 1994] M. Gendreau, A. Hertz, G. Laporte. (1994). A tabu search heuristic for the vehicle routing problem. Management Science 40, 1994, pp. 1276–1290.

[Gendreau et al., 1998] M. Gendreau, A. Hertz, G. Laporte, M. Stan. (1998). A generalized insertion heuristic for the traveling salesman problem with time windows, Operations Research, Vol. 43, 1998, pp. 330–335.

[Gill, 1958] S. Gill. (1958). Paralell programming. The computer journal,1: 2-1.

[Glover y Laguna, 1997] F. Glover, M. Laguna. (1997). Tabu Search.University of Colorado at Boulder.Hardbound, ISBN 0-7923-9965-X Paperback, ISBN 0-7923-8187-4.July 1997, 408 pp.

[González y Aristizábal, 2006] G. González, F. Aristizábal. (2006). Metaheuristics applied to vehicle routing. A case study: Parte 1: formulating the problem. Ing. Investig., set./dez. 2006, vol.26, no.3, p.149-156. ISSN 0120-5609.

[González y Aristizabal, 2007] Guillermo González Vargas / Felipe González Aristizábal. (2007). Meta heurísticas aplicadas al ruteo de vehículos un caso d estudio parte 2. Algoritmo genético. Comparación con una solución heurística Ingeniería e Investigación, abril, año/vol. 27, número 001.Universidad Nacional de Colombia. Bogotá, Colombia. p. 149-157

[Gupta. y Krishnamurti, 1997] A. Gupta., R. Krishnamurti, R. (1997). Parallel algorithms for vehicle routing problems. Fourth International Conference on High-Performance Computing, pp.144-151.

[Halse, 1992] K. Halse. (1992). Modeling and Solving Complex Vehicle Routing Problems, PhD Thesis, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, Lyngby, 1992.

[Hartmanis y Hopcroft, 1971] J. Hartmanis J. E. Hopcroft. An Overview of the Theory of Computational Complexity Journal of the ACM (JACM) archiveVolume 18 , Issue 3 (July 1971) table of contentsPages: 444 - 475 . Year of Publication: 1971.ISSN:0004-5411. Publisher ACM New York, NY, USA

[Hermosilla y Barán, 2005] A. Hermosilla, B. Barán. (). Comparación de un sistema de colonias de hormigas y una estrategia evolutiva para un Problema Multiobjetivo de Ruteo de Vehículos con Ventanas de Tiempo. s/f., Disponible en: http://www.cnc.una.py/invest/paper2/augCLEI.pdf. Consultado en: Febrero de 2005.

[Hillier y Lieberman, 1991] F. S. Hiller, G.J. Lieberman. (1991). IO Análisis de sensibilidad
http://www.elprisma.com/apuntes/matematicas/analisisdesensibilidad/default6.asp.

[Hillier y Lieberman, 1991 b]F. S. Hillier, G. J. Lieberman. (1991). “Introducción a la investigación de operaciones”, Quinta edición, McGRAW-HILL, México. ISBN 968-422-993-3

[Holland, 1962 b] J. Holland, (1962). Concerning efficient adaptive systems. In M. C. Yovits, G. T.Jacobi, and G. D. Goldstein, editors, Self-Organizing Systems—1962, pages 215–230. Spartan Books, Washington, D.C.

[Holland, 1962] J. Holland. (1962). Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery, 9:297–314, 1962.

[Holland, 1967] J. Holland. (1967). A universal computer capable of executing an arbitrary number of subprograms simultaneosly. In proceedings of east joint computer conference, pp 8-113.

[Holland, 1975] J. Holland. (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan, 1975.

[Homberger y Gehring, 1999] J. Homberger, H. Gehring. (1999). "Two evolutionary metaheuristics for the vehicle routing problem with time windows". Indformation systems nd Operationl Research, 37,297-318.

[Hopcroft y Ullman, 1993] J. Hopcroft, J. Ullman. (1993).Introducción a la teoría de autómatas. Edit. CECSA 1993.

[Joyanes, 2000] JOYANES AGUILAR, Luis. (2000). Programación en C++ algoritmos, Estructuras de Datos y Objetos. España, Ed. McGraw-Hill. 2000.

[Karp, 1972] R.M. Karp. (1972):Reducibility among combinatorial problems. In complexity of computer computations. R.E. Miller and J.W. Thatcher, Eds. Plenum Press, New York (1972) 85-104.

[Kirkpatrick et al., 1983] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. (1983). Optimization by simulated annealing. Science, 220:671-680, 1983.

[Kohl et al., 1997] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, F. Soumis (1997). K-PathCuts for the Vehicle Routing Problem with Time Windows, Technical Report IMMREP-1997-12, Technical University of Denmark

[Koza, 1992] J. R. Koza. (1992). Genetic Programming. On the Programming of Computers by Means of natural Selection. The MIT Press.

[Lahoz-Beltra, 2005] R. Lahoz-Beltra. (2005). Bioinformatica- simulacion vida artificial e inteligencia artificial: simulación, vida artificial e inteligencia

artificial Autor Rafael Lahoz-Beltra Publicado por Ediciones Díaz de Santos, 2005 ISBN 8479786450, 9788479786458 574 páginas

[Larsen, 2001] J. Larsen. (2001). Parallelization of the vehicle routing problem with time windows. Department of Computer Science (DIKU) at the University of Copenhagen,Technical University of Denmark (DTU).ISSN 0909-3192. 2001

[Lee et al., 2002] Ch. Lee, M. Epelman, C. Chelsea, Y. Bozer. (2002). A Shortest Path Approach to the Multiple-Vehicle Routing Problem with Split Pick-Ups, 2002., Disponible en: http://osiris.tuwien.ac.at/~wgarn/VehicleRouting/mvrpsp_ts.pdf, Consultado en: Febrero de 2005.

[Lenstra y Rinnoy, 1981] J. K. Lenstra, A. H. G, K. Rinnooy. (1981). "Complexity of Vehicle Routing and Scheduling Problems," Networks, Vol. 11, No. 2, pp. 221-227.

[Louis et al., 1999] S. Louis, X. Yin, Z. Yuan. (1999). Multiple Vehicle Routing with Time Windows Using Genetic Algorithms., Memorias de Proceedings of the Congress of Evolutionary Computation, 1999, pp. 1804-1808.

[Maeda et al., 1999] O. Maeda, M. Nakamura, B. M. Ombuky, K. Onaga. (1999). A genetic algorithm approach to vehicle routing problem with time deadlines in geographical information systems. International Conference on Systems, Manand Cybernetics. Vol. 4, 1999, pp.595-600.

[Maimon y Rokach, 2005] O. Maimon, L. Rokach. (2005). Data Mining And Knowledge Discovery Handbook. Speinger 2005.

[Martí y Cunquero, 2003] R. Martí, R. Cunquero. (2003). “Algoritmos heurísticos en optimización combinatoria”, departamento de estadística e investigación operativa, facultad de ciencias matemáticas. Universidad de valencia

[Martí, 2003] R. Martí. (2003) Procedimientos Metaheurísticos en Optimización Combinatoria, Matemàtiques 1(1), 3-62 R. Martí (2003).

[Martínez-Morales, 2006] A. A. Martínez-Morales. "Algoritmo Basado en Tabu Search para el Cálculo del Índice de Transmisión de un Grafo". Departamento de computación Facultad de ciencias y tecnología. FARAUTE

de Ciencias y Tecnología, Vol. 1, No. 1, páginas 31-39. Universidad de Carabobo, Valencia, Estado Carabobo, Venezuela (2006).

[McQueen, 1967] J. McQueen. (1967). Some methods for classification and analysis of multivariate observations, 5-th Berkeley Symposium on mathematics, Statistics and Probability, 1, S. 281-298.

[Min, 1989] H. Min. (1989) The multiple vehicle routing problem with simultaneous delivery and pick-up points. Transportation Research. v23A. 377-386.

[Nesmachnow, 2004] S. Nesmachnow. (2004). "algoritmos genéticos paralelos y su aplicación al diseño de redes de comunicaciones confiables" Instituto de computación facultad de Ingeniería. Universidad de la república. Montevideo Uruguay(2004)

[Olatz y Gallego, 200] A. Olatz, A. Gallego. (2002). Soluciones basadas en simulated annealing para el VRPTW. Universidad del pais Vasco,Donostia.

[Olivera, 2004] A. Olivera (2004). Heurísticas para problemas de ruteo de vehículos, Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay. 2004. Disponible en: http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0408.pdf. Consultado en Febrero de 2005.

[Ombuki et al., 2004] B. Ombuki, B. J. Ross, and F. Hanshar. (2004).Multi-objective Genetic Algorithms for Vehicle Routing Problem with Time Windows.Technical Report # CS-04-02 January 2004. Brock University Department of Computer Science. St. Catharines, Ontario Canada L2S 3A1. www.cosc.brocku.ca

[Pacheco et al., 2000] J. Pacheco, A. Aragón, C. Delgado. (2000). “Diseño de Algoritmos para el problema del Transporte Escolar. Aplicación en la Provincia de Burgos”. Questiió, 1, 24, pp. 55-82.

[Pacheco y Delgado, 2000] Joaquín A. Pacheco y Cristina R. Delgado. (2000). Resultados de diferentes experiencias con búsqueda local aplicadas a problemas de rutas. Universidad de BURGOS. Dpto. Economía Aplicada. Fac. CC EE y EE. C/Francisco Vitoria s/n BURGOS 09006 e-mail: jpacheco@ubu.es cdelgado@ubu.es

[Pacheco y Delgado, 2001] J. Pacheco, C. Delgado. (2001). "Uso de Conjunto de Concentración en Búsqueda Tabú para problemas de rutas". Rect@, ASEPUMA, vol. 3, nº 1, pgs. 49-87, (2001).

[Pacheco y Marti, 2006] J. Pacheco, R. Marti. (2006). “Tabu Search for a Multi-Objective Routing Problem” Journal of Operations Research Society, 57, pp. 29-37 (2006)

[Papadimitriou y Steiglitz, 1998] C.H. Papadimitriou, K. Steiglitz. (1998). “Combinatorial optimization. Algorithms and complexity” Dover publication, Inc. Mineola N.Y. ISBN 0-486-40258-4

[Potvin et al., 1996] J. Potvin, J.Y.,T.Kervahut, B.L.Garcia, J.M. Rousseau. (1996). The vehicle routing problem with time windows; partI: tabu search. INFORMS journal on computing 8, 158-164 (1996)

[Potvin y Bengio, 1996] J. Y. Potvin, S. Bengio. (1996).The Vehicle Routing Problem with Time Windows - Part II:Genetic Search. CiteSeerPSU:481594, oai:CiteSeerPSU:531140

[Pressman, 2006] R. S. Pressman. (2006). Ingeniería del Software. Un enfoque práctico. 6ª ed. 2006: Mc Graw Hill, Interamericana de España S.A.U.

[Reid et al., 1991] S. Reid, J. Principe, C. Lefevre. (1991). Solution real problem Neurodimension. Gainesville, Florida.1991 http://www.nd.com/products/genetic/selection.htm

[Sanvicente-Sánchez y Frausto-Solís, 2004] Sanvicente-Sánchez, H., Frausto-Solís, J.: A Method to Establish the CoolingScheme in Simulated Annealing Like Algorithms. In: Proceedings of the International Conference on Computational Science and Its Applications -ICCSA 2004. Part III-, Assisi, Italy, May 14-17. (2004) 755-763.

[Savelsbergh, 1985] M. W.P Savelsbergh. (1985). Local search for routings problems with time windows. Transportation Science 29(2), pp.156-166.1985

[Seyed, 1999] H. Seyed. (1999). Parallel processing and parallel algorithm. Theory and computation. Roosta, editor. Springer-Verlag.

[Solomon, 1985] M. M. Solomon. (1985). Algorithms for Vehicle Routing and Scheduling Problems with time window constraints. Northeastern University, Boston, Massachusetts, December 1985.

[Solomon, 1987] M. M. Solomon. (1987). Algorithms for Vehicle Routing and Scheduling Problems with Time Window Constrains. Operations Research 35 (1987)

[Syswerda, 1991] G. Syswerda (1991). Schedule optimization using genetic algorithms. L. Davis (ed.). Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 332-349.

[Tan et al., 2000] K. C. Tan, L.H Lee, Q. L. Zhu, K. Ou. (2000). Heuristic Methods for Vehicle Routing Problem with Time Windows, Artificial Intelligent in Engineering, pp. 281-295.

[Tan et al., 2001] K. C. Tan, T.H. Lee, K. Ou, L.H. Lee. (2001). A messy genetic algorithm for the vehicle routing problem with time window constraints. Congress on Evolutionary Computation, Vol. 1, 2001, pp.679-686.

[Tansini et al., 2008] L. Tansini, M. Urquhart, O. Viera. (s/f). Comparing assignment algorithms for the multi-depot VRP Diponible en: http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0108.pdf. Consultado en: Julio 2008

[Thangiah et al., 1991] S.R. Thangiah, E.N. Kendall and P. L. Juell (1991). GIDEON: A Genetic Algorithm System for Vehicle Routing Problems with Time Windows. Proceedings of the Seventh IEEE Conference on Artificial Intelligence Applications, Miami, Florida.

[Thangiah, 1995] Sam R. Thangiah.(1995). Vehicle Routing with Time Windows using Genetic Algorithms.Submitted to the book on "Applications Handbook of Genetic Algorithms:New Frontiers&quot 1; Artificial Intelligence and Robotics Laboratory. Computer Science Department Slippery Rock University Slippery Rock PA 16057 U.S.A.

[Thangiah, 1999] S.R. Thangiah. (1999) "Hybrid Genetic Algorithm,Simulated Annealing and Tabu Search Heuristic for Vehicle Routing Problems with Time Windows," Practical Handbook of Genetic Algorithms: Complex CodingSystems, Vol III, L. Chambers (Ed), CRC Press, Florida.

[Thompson y Psaraftis, 1989] P.M. Thompson, H. Psaraftis. (1989). Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems. Operations Research, 41, 935-946.

[Toth y Vigo, 2001] P. Toth, D. Vigo. (2001). The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications. Society of Industrial and Applied Mathematics. Philadelphia. USA (2001)

[Toth y Vigo, 2002] P. Toth, D. Vigo. (2002). The Vehicle Routing Problem. SIAM, ISBN 0-89871-579-2

[Vacic, 2002] V. Vacic. (2002). Vehicle Routing Problem with Time Windows; 2002., Disponible en: http://www.bridgeport.edu/sed/projects/cs399/vacic/vrptw.html. Consultado en: Febrero 2005

[Villagra et al., 2007] A. Villagra, D. Pandolfi. G. Leguizamón. (2007). "Selección de centroides para algoritmos de clustering a través de técnicas metaheuristicas". Universidad Nacional de la patagonia Austral. Argentina. XIII Congreso Argentino de Ciencias de la Computación -- CACIC 2007. Universidad Nacional del Nordeste, Corrientes. Universidad Tecnologica Nacional, Resistencia Octubre 2007. pp 1773 - 1783. ISBN 978-950-656-109-3

[Volkan, 2005] A. Volkan. (2005). A GA Based Meta-Heuristic for the Capacitated Vehicle Routing Problem with Simultaneous Pick-up and Deliveries. Disponible en: http://www.msie.sabanciuniv.edu/thesis/arif_volkan_vural.ppt. Consultado en: Marzo 2005

[Wagner y Affenzeller, 2004] S. Wagner, M. Affenzeller. (2004). “The HeuristicLab Optimization Environment”, Technical Report. Institute of Formal Models and Verification, Johannes Kepler University Linz, Austria. 2004.

[Wagner y Affenzeller, 2005] S. Wagner, M. Affenzeller. (2005). “SexualGA: Gender-Specifc Selection for Genetic Algorithms”, Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI) 2005, vol. 4, pp. 76-81, 2005.

[Wetzel, 1983] A. Wetzel, (1983). Evaluation of the effectiveness of genetic algorithms in combinational optimization. University of Pittsburgh, Pittsburgh (unpublished).

[Whitley, 1993] D. Whitley. (1993). A genetic Algorithm Tutorial, Tech. Report CS-93-103, Colorado State University.

[Zhu, 2000] K. Zhu . (2000). A new genetic algorithm for VRPTW., Presented at IC-AI 2000, Las Vegas, USA, 2000.

APENDICE A

CÁLCULO DE COMPLEJIDAD DE ALGORITMOS.

ANEXO1 CAPITULO I . Sección 1.2

Reglas y estimaciones para calcular el polinomio g(n).

El siguiente paso para entender como calcular la complejidad es saber cómo obtener el polinomio g(n). Par obtener el polinomio en base al análisis del pseudocódigo se define las siguientes reglas para la notación asintótica:

1.- Regla de la suma

Si T1(n) y T2(n) son las funciones que expresan los tiempos de ejecución de dos fragmentos de un programa, y se acotan de forma que se tiene:

$T1(n) = O(f1(n))$ y $T2(n) = O(f2(n))$

Se puede decir que:

$T1(n) + T2(n) = O(\max(f1(n), f2(n)))$

2.- Regla del producto

Si T1(n) y T2(n) son las funciones que expresan los tiempos de ejecución de dos fragmentos de un programa, y se acotan de forma que se tiene:

$T1(n) = O(f1(n))$ y $T2(n)=O(f2(n))$

Se puede decir que:

$T1(n)\ T2(n) = O(f1(n)\ f2(n))$

Etimaciones definidas por Joyanes [Joyanes 1996] para algoritmos no recursivos.

1.- Asignaciones y expresiones simples (=)

El tiempo de ejecución de toda instrucción de asignación simple, de la evaluación de una expresión formada por términos simples o de toda constante es O(1).

2.- Secuencias de instrucciones (;)

El tiempo de ejecución de una secuencia de instrucciones es igual a la suma de sus tiempos de ejecución individuales. Para una secuencia de dos instrucciones S1 y S2 el tiempo de ejecución está dado por la suma de los tiempos de ejecución de S1 y S2:

$T(S1 ; S2) = T(S1) + T(S2)$

Aplicando la regla de la suma:

$O(T(S1 ; S2)) = \max(O(T(S1), T(S2)))$

3.- Instrucciones condicionales (IF, SWITCH-CASE)

El tiempo de ejecución para una instrucción condicional IF-THEN es el tiempo necesario para evaluar la condición, más el requerido para el conjunto de instrucciones que se ejecutan cuando se cumple la condición lógica.

$T(\text{IF-THEN}) = T(\text{condición}) + T(\text{rama THEN})$

Aplicando la regla de la suma:

$O(T(\text{IF-THEN})) = \max(O(T(\text{condición}), T(\text{rama THEN})))$

El tiempo de ejecución para una instrucción condicional de tipo IF-THEN-ELSE resulta de evaluar la condición, más el máximo valor del conjunto de instrucciones de las ramas THEN y ELSE.

$T(\text{IF-THEN-ELSE}) = T(\text{condición}) + \max(T(\text{rama THEN}), T(\text{rama ELSE}))$

Aplicando la regla de la suma, su orden está dada por la siguiente expresión:

$O(T(\text{IF-THEN-ELSE})) = O(T(\text{condición})) + \max(O(T(\text{rama THEN})), O(T(\text{rama ELSE})))$

Aplicando la regla de la suma:

$$O(T(\text{IF-THEN-ELSE})) = \max(O(T(\text{condición})), \max(O(T(\text{rama THEN})), O(T(\text{rama ELSE}))))$$

El tiempo de ejecución de un condicional múltiple (SWITCH-CASE) es el tiempo necesario para evaluar la condición, más el mayor de los tiempos de las secuencias a ejecutar en cada valor condicional.

4.- Instrucciones de iteración (FOR, WHILE-DO, DO-WHILE)

El tiempo de ejecución de un bucle FOR es el producto del número de iteraciones por la complejidad de las instrucciones del cuerpo del mismo bucle.

Para los ciclos del tipo WHILE-DO y DO-WHILE se sigue la regla anterior, pero se considera la evaluación del número de iteraciones para el peor caso posible. Si existen ciclos anidados, se realiza el análisis de adentro hacia fuera, considerando el tiempo de ejecución de un ciclo interior y la suma del resto de proposiciones como el tiempo de ejecución de una iteración del ciclo exterior.

5.- Llamadas a procedimientos

El tiempo de ejecución está dado por, el tiempo requerido para ejecutar el cuerpo del procedimiento llamado. Si un procedimiento hace llamadas a otros procedimientos “no recursivos”, es posible calcular el tiempo de ejecución de cada procedimiento llamado, uno a la vez, partiendo de aquellos que no llaman a ninguno.

- Ejemplo: función no recursiva que halla el factorial de un número n cualquiera

```
int factorial(int n)                          O(1)
{
        int fact = 1;                         O(1)
        for(int i = n; i > 0; i--)            O(n)
```

```
        fact = fact * i;                                O(1)
        return fact;                                    O(1)
    }
```

Su complejidad es lineal O(n), debido a que se tiene un bucle FOR cuyo número de iteraciones es n.

- Ejemplo: Si el bucle se repite un número fijo de veces, liberado de n, entonces el bucle introduce una constante que puede ser absorbida.

```
        int y, z, k = 10;                               O(1)
        for(int i = 0; i < k; i++)                      k * O(1)
        {
            y = y + i;                                  O(1)
            z = z + k;                                  O(1)
        }
```

Su complejidad es constante; es decir, O(1), debido a que se tiene un bucle for independiente de n.

- Ejemplo: Dos bucles anidados dependientes de n.

```
    for(int i = 0; i < n; i++)                          O(n)
    {
        for(int z = 0; z < n; z++)                      O(n)
        {
                if(vector[z] > vector[z + 1])           O(1)
                {
                aux = vector[z];                        O(1)
                vector[z] = vector[z + 1];              O(1)
                vector[z + 1] = aux;                    O(1)
                }
        }
    }
```

Tenemos O(n) * O(n) * O(1) = O(n2), complejidad cuadrática.

- Ejemplo: Dos bucles anidados, uno dependiente n y otro dependiente del bucle superior.

```
for(int i = 0; i < n; i++)                          O(n)
{
        for(int z = n; z < i; z--)                  O(n)
        {
                if(vector[z] < vector[z - 1])       O(1)
                {
                aux = vector[z];                            O(1)
                vector[z] = vector[z - 1];          O(1)
                vector[z - 1] = aux;                O(1)
                }
        }
}
```

Tenemos que el bucle exterior se ejecuta n veces $\Rightarrow$ O(n), el bucle interno se ejecuta n + … + 3 + 2 + 1 veces respectivamente, o sea $(n * (n+1))/2 \Rightarrow$ O(n).

O(n) * O(n) * O(1) = O(n2), complejidad cuadrática.

- Ejemplo: Bucles donde la evolución de la variable de control es ascendente no lineal.

```
int c = 1;                                                  O(1)
while(c < n)                                        O(log n)
{
        if(vector[c] < vector[n])                   O(1)
                {
                aux = vector[n];                            O(1)
                vector[n] = vector[c];              O(1)
                vector[c] = aux;                            O(1)
```

```
                    }
                    c = c * 2;
}
```

Para este ejemplo al principio el valor de c es 1, al cabo de x iteraciones será 2x $\Rightarrow$ el número de iteraciones es tal que n $\leq$ 2x donde x es el entero inmediato superior de n; x = log2 n iteraciones, $\Rightarrow$ para este caso es:
O(1) * O(log n) * O(1) = O(log n), complejidad logarítmica.

- Ejemplo: Bucles donde la evolución de la variable de control es descendente no lineal.

```
int c = n;                                          O(1)
while(c > 1)                                        O(log n)
{
                    vector[c] = c;                  O(1)
          c = c / 2;                                O(1)
}
```

Para este ejemplo al principio el valor de c es igual a n, al cabo de x iteraciones será n*2-x $\Rightarrow$ el número de iteraciones es tal que n*2-x $\leq$ n; un razonamiento análogo nos lleva a log2 n iteraciones, $\Rightarrow$ para este caso es:
O(1) * O(log n) * O(1) = O(log n), complejidad logarítmica.

- Ejemplo: Bucles donde la evolución de la variable de control no es lineal, junto a bucles con evolución de variable lineal.

```
int c, x;                                           O(1)
for(int i= 0; i < n; i++)                           O(n)
{
c = i;                                              O(1)
          while(c > 0)                              O(log n)
```

```
        {
            x = x % c;                    O(1)
            c = c / 2;                    O(1)
        }
        x = x + 2;                        O(1)
}
```

Tenemos un bucle interno de orden O(log n) que se ejecuta O(log n) veces, luego el conjunto de ordenes es de orden:
O(1) * O(n) * O(log n) = O(n log n), complejidad cuasi-lineal.

APÉNDICE B

CÓDIGO PARA EL CÁLCULO DE LA COMPLEJIDAD DEL ALGORITMO GA-VRPTW-SN

```
01 int main()
02{
03 printf("Archivo de Prueba: ");
04 scanf("%s", archivo);
05 ifstream ent(archivo);
06    while(!n )
07     {
08       for (k=0 to n)  /*MATRIZ en CEROS*/
09        {
10         for (j=0 to n){OriginalFinal[k][j].dato=0;}
11         }
12       for (k=0 to n)/*DISTANCIAS,llena matriz*/
13        {
14         for (j=0 to n)
15          {
16            if(k==j)
17             {
18               OriginalFinal[k][j].dato=0.0;
19               }
20             else
21             {
22              if (OriginalFinal[k][j].dato==0)
23               {
24                 a=instancia.XCOOR[j]- instancia.XCOOR[k];
25                 b=instancia.YCOOR[j]-instancia.YCOOR[k];
26                OriginalFinal[k][j].dato=sqrt((a*a)+(b*b));
27                }
28              }
29          }// fin for j
30        }// fin for k
31       lee archivo población inicial; /*comienza selección*/
32       while(ga!=generaciones=20)  //criterio de paro
33       {
34         printf("\nGeneracion: %d tiempiniGen:",ga,startGA);
```

```
35          for(i=0 to m=1000)
36            {
37              for(j=0 to m=1000)
38              {
39                if(MT[i][0].total< MT[j][0].total)
40                 {
41                   for(x=0 to m=1000)
42                     {
43                       selecciona mejor()
44                     }
45                 }
46               }// fin for j
47            }// fin for i
48            for(iii=0 to m=1000pob) /*inicia crosover*/
49            {
50              for(i=0 to n)
51                 {
52                   Checa restricciones no repetición de nodos()
53                 }
54                Realiza cruzamiento()
55              }//fin del for iii crossover koko
56            for(iii=0 to m=1000) /*inicia VECINDAD*/
57             {
58               for(z=0 to 20 iteraciones)
59                {
60                 for(x=0 to 10 vecinos)//gen n vecino
61                  {
62                    for(i=0 to n)
63                    {
64                      copia individuo inicial()
65                    }
66                    for(i=0 to n)
67                    {
68                       localiza nodos para generar vecino()
69                    }
70                    for(i=0 to n)  //checa restricciones
71                    {
72                      Checa vehiculo,demanda
73                    } fin del for
```

```
74                  for(i=0 to n)
75                  {
76                   checa restricciones de ventana
77                  } //del for
78                  if(vecindare==0) //si cumple
79                  {
80                    for(i=0 to n)//copia indiv. Orig.
81                     {
82                       copia la solución actual()
83                     }//cambia nodo RV1 y RV2
84                     genera vecino()
85                    }///vecindare=0
86               }//for n vecinos
87               for(iy=0 to 10 vecinos)
88               {
89                for(x=0 to n)
90                  {
91                    calcula distancias()
92                  }//for x
93                }//for iy
94               for(x=0 to 10 vecionos)
95               {
96                 busca mejor vecino()
97               }
98               if(encontromenor==1)
99               {//asigna el mejor de los N vecinos a la población
100               for(i=0 to m=1000)
101               {
102                   mete mejor a sig generación()
103                }
104               } //fin si de encontromenor=1
105          }//fin del for z iteraciones
106      } //for iii individuos (fase vecindad)
107    }//end del while   GA
108  }//fin del while ent.eof
109 printf("\n\n\tFin del proceso. Presione tecla para continuar");
110 getch();return(0);
111}// fin main()
```

APÉNDICE C

CÓDIGO PARA ELCÁLCULO DE LA COMPLEJIDAD DEL ALGORITMO GA-RVPTW-PN

A continuación se presenta el algoritmo genético paralelo.

```
Algoritmo paralelo GA-VRPTW-VECINDAD
01 inicio
02{
03    while(!n )
04     {
05       for (k=0 to n)
06         {
07           for (j=0 to n){llena matriz con ceros}
08         }
09       for (k=0 to n)/*DISTANCIAS,llena matriz*/
10        {
11          for (j=0 to n)
12          {
13            Llena matriz de distancias
14          }// fin for j
15        }// fin for k
16       while(ga!=NUMgeneraciones=20)  //criterio de paro
17       {
18         for(i=0 to TAMpoblacion)
19          {
20            for(j=0 to TAMpoblacion)
21            {
22              Torneo
23            }// fin for j
24          }// fin for i
25         for(iii=0 to TAMpoblacion) /*inicia crosover*/
26          {
27            for(i=0 to n)
28               {
29                 Checa restricciones no repetición de nodos
30               }
31               Realiza cruzamiento
32          }//fin del for iii crossover koko
```

```
    /***************INICIA REGION PARALELA*************************/
33  #pragma omp parallel shared(Numvecinos,TAMpoblacion, chunk_size)
34  {
35    #pragma omp for schedule(dynamic, chunk_size) nowait
36         for(iii=0 to TAMpoblacion) /*inicia VECINDAD*/
37           {
38              for(z=0 to NUMiteraciones)
39              {
40                 for(x=0 to NUMvecinos)//gen n vecino
41                 {
42                   Genera vecinos
43                 }//for n vecinos
44                for(iy=0 to NUMvecinos)
45                {
46                 for(x=0 to n)
47                  {
48                    calcula distancias
49                  }//for x
50                }//for iy
51               for(x=0 to NUMvecinos)
52               {
53                 busca mejor vecino
54               }
55               if(encontromenor==1)
56               {//asigna el mejor de los N vecinos a la población
57                for(i=0 to TAMpoblacion)
58                 {
59                     mete mejor a siguiente población
60                 }
61                } //fin si de encontromenor=1
62         }//fin del for z iteraciones
63      } //for iii individuos (fase vecindad)
64  }
   /*********************************FIN REGION PARALELA***************/
65    }//end del while   GA
66  }//fin del while ent.eof
67 }// fin main()
```

APÉNDICE D

CÓDIGO DEL ALGORITMO GA-VRPTW-SN
ANEXO 1 ALGORITMO SECUENCIAL

```
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include<malloc.h>
#include <ctype.h>
#include <values.h>
#define N2 10
#define chunck_size 5
char archivo[50];
int tamanonodo=0,nodo;
FILE *f = NULL;//filefile
char name[5];
int CUENTA,iiiiii=0,jjjjjj=0,rrr=1;
int INDIVIDUO;
int NUMNOD=101,CAPACIDADVEHI=200;
float
tiempoCalculadoGA,tiempoCalculadoCross,tiempoCalculadoVecino,tiempoCalculado,tiempo
CalculadoBest,tiempoCalculadopob;
double Tinicio=0,Tfin,tiempoI,tiempoF;
double mascortaPARNODO(int I,int F);
int casosCSP(int NumParejas);
int calculaNODOINCONSISTENTE();
void algoritmogeneticoSIMPLE();
double MASCORTA();
double VENTANAS();
void TIMEWINDOWS();
void divideinstancia();
void algoritmogeneticoBPP();
void algoritmogeneticomtsp();
void MATDistancias();
void genpobinialgo0();
void calculadisttotal();
void algoritmogeneticoKOKO();
int combinaPARES();
double CALCULARUTAS();
void TIMEWINDOWS();
void rutakmeans(void);
void Permutaciones(int cad[2000], int I);
int CNN[101];
float OPTIMOKNOW;
int individuos=0,generaciones=0,vecinos=0,iteraciones=0;
void ordenaVEC();
int omp_get_num_threads(void);
int omp_get_max_threads(void);
int omp_get_thread_num(void);
```

```
int omp_get_num_procs(void);
void omp_set_dynamic(int dynamic_threads);
int omp_get_dynamic(void);
int omp_in_parallel(void);
void omp_set_nested(int nested);
int omp_get_nested(void);

struct  datosinstancia{
   char instancia[5];
        int VN;
        int     C;
        int     CN[101];
        int     XCOOR[101];
        int     YCOOR[101];
        int     DEMAND[101];
        int     READYTIME[101];
        int     DUEDATE[101];
        int     SERVICETIME[101];
 }instancia;

//estructura que guarda resultados
struct  Resultados{
   int Norigen;
   int Ndestino;
   double valor;
        float DemNodo;
        int vh;
        int tiempo;
}Resultado[1001];

//estructura que guarda las distancias
struct  datos{
   double dato;
   int marca;
}Distancias[101][101],Original[101][101],OriginalFinal[101][101],DistanciaNueva[101][101],T
empoDistancias[101][101];

struct QQ{
        int nodo;
        int vh;
        int dem;
        int tiempo;
}R[1001], RAUX[50];

struct QA0
{
        int nodo;
        int vh;
        int dem;
        int tiempo;
        double total;
        double tiempoI;
        double tiempoF;
```

```
        int nitera;
}MT[2001][1000];

struct QA1
{
        int nodo;
        int vh;
        int dem;
        int tiempo;
        double total;
        double tiempoI;
        double tiempoF;
        int nitera;
}MTAUX[2001];

struct QA2
{
        int nodo;
        int vh;
        int dem;
        int tiempo;
        double total;
        double tiempoI;
        double tiempoF;
        int nitera;
}MTVEC[10000][100];

struct VENTANAS{
   int Nodo;
   int E;
   int LTServicio;
   int DELTAR;
   int ele;
}TablaVENTANA[1001];


int main(int argc,char *argv[])
{   float a[N2], b[N2], total[N2]; int i;
    int cuenta=0,NumCasos=1,ZeroNodo=0,bandera=1;
    float DemandaT=0;
    double DISTindividuoNuevo=0;
    int factible=0,nofactible=0,ham=0;
    int BANDERAPOB=0;
    char cadena[300];
    cadena[0]='\0'; FILE  *archDatos;
    clock_t start, end;
    int repetir;
    srand((unsigned)time(NULL));
    sprintf(name,"resultados-GAx.txt");
    if(!(f=fopen(name,"w"))){printf("error al abrir el archivo");return 0;}
    printf("Proporcione el Archivo de la instancia a resolver: ");
    scanf("%s", archivo);
    if((archDatos=fopen(archivo,"rt"))==NULL)
```

```
    { printf(" No existe el archivo de datos ==> [%s]%c",archivo,7);}
    printf("\n       El archivo es %s", archivo);
    fprintf(f,"\n    El archivo es %s", archivo);
    fgets(instancia.instancia,5,archDatos);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
      fgets(cadena,300,archDatos);
      fgets(cadena,300,archDatos);
      fscanf(archDatos,"%d\t",&instancia.VN);
      fscanf(archDatos,"%d\t",&instancia.C);
      fgets(cadena,300,archDatos);
      fgets(cadena,300,archDatos);
      fgets(cadena,300,archDatos);

      for(repetir=0;repetir<101;repetir++)   //41
      {
        fscanf(archDatos,"%d\t",&instancia.CN[repetir]);
        fscanf(archDatos,"%d\t",&instancia.XCOOR[repetir]);
        fscanf(archDatos,"%d\t",&instancia.YCOOR[repetir]);
        fscanf(archDatos,"%d\t",&instancia.DEMAND[repetir]);
        fscanf(archDatos,"%d\t",&instancia.READYTIME[repetir]);
        fscanf(archDatos,"%d\t",&instancia.DUEDATE[repetir]);
        fscanf(archDatos,"%d\t",&instancia.SERVICETIME[repetir]);
      }
      tamanonodo=repetir;
      MATDistancias();
      algoritmogeneticoKOKO();
      fclose(archDatos);
      fclose(f);
      printf("\n\n\tFin.");
      return(0);
}


void MATDistancias()
{
   float a,b,ra,rb,rc; int k=0,j=0,i=0;
   for (k=0;k<tamanonodo-2;k++)     /*LLENADO DE MATRIZ CON CEROS*/
   { for (j=0;j<tamanonodo-2;j++) {OriginalFinal[k][j].dato=0;}
   }
   for (k=0;k<tamanonodo-1;k++)  /*CALCULA DISTANCIAS y llena matriz*/
   {
    for (j=0;j<tamanonodo-1;j++)
    {
          if(k==j)
          {
            OriginalFinal[k][j].dato=0.0;
          }
          else
          {  if (OriginalFinal[k][j].dato==0)   //Distancias[i][j].dato==0)
            {
              a=instancia.XCOOR[j]- instancia.XCOOR[k];
              b=instancia.YCOOR[j]-instancia.YCOOR[k];
```

```
              OriginalFinal[k][j].dato=sqrt((a*a)+(b*b));
            }
          }
    }
  }
}

void algoritmogeneticoKOKO()
{ int ga=0,i=0,j=0,x=0,iii=0,R1,R2,tempR1,tempR2,RV1,RV2,tempRV1,tempRV2,nitera=0;
  double DisT=0.0;
  clock_t startGA, endGA,startBest,endBest,startCross,endCross,startVecino,endVecino;
  clock_t startpob,endpob;
  printf("\nEmpezo el algoritmo genetico simple propuesto por koko");
  printf("\n(Seleccion=Best, Crossover=koko, Muta=VECINDAD");
  printf("\nPorfavor introduzca el optimo de la instancia(827.3): ");
  scanf("%g",&OPTIMOKNOW);
  printf("\nPorfavor introduzca el numero de individuos(1000): ");
  scanf("%i",&individuos);
  printf("\nPorfavor introduzca el numero de generaciones (20): ");
  scanf("%i",&generaciones);
  printf("\nPorfavor introduzca el numero de vecinos(10): ");
  scanf("%i",&vecinos);
  printf("\nPorfavor introduzca el numero de iteraciones(9900): ");
  scanf("%i",&iteraciones);
  startpob=clock();
  genpobinialgo0();
  endpob=clock();
  startGA=clock();

  while(ga!=generaciones)  //criterio de paro
  { printf("."); ga++;
  printf("\nGeneracion: %i",ga);
  //////SECCION BEST
  startBest=clock();
  for(i=0;i<individuos;i++)
          {
          for(j=0;j<individuos;j++)
          {
          if(MT[i][0].total< MT[j][0].total)
          {
            for(x=0;x<tamanonodo-1;x++)
                {
                    MTAUX[x].dem=MT[i][x].dem;
                    MTAUX[x].nodo=MT[i][x].nodo;
                    MTAUX[x].tiempo=MT[i][x].tiempo;
                    MTAUX[x].vh=MT[i][x].vh;
                    if (x==0)
                    {
                    MTAUX[0].total=MT[i][0].total;
            }
            MT[i][x].dem=MT[j][x].dem;
            MT[i][x].nodo=MT[j][x].nodo;
            MT[i][x].tiempo=MT[j][x].tiempo;
```

```
          MT[i][x].vh=MT[j][x].vh;
          if (x==0)
         {
          MT[i][0].total=MT[j][0].total;
         }
          MT[j][x].dem=MTAUX[x].dem;
          MT[j][x].nodo=MTAUX[x].nodo;
          MT[j][x].tiempo=MTAUX[x].tiempo;
           MT[j][x].vh=MTAUX[x].vh;
           if (x==0)
           {
                  MT[j][0].total=MTAUX[0].total;
                  MTAUX[x].total=0;
            }
           }
       }
       }
       }
       endBest=clock();
 tiempoCalculadoBest = tiempoCalculadoBest +((endBest - startBest) /
        (CLOCKS_PER_SEC * 1.0f));
 printf(".");
////SECCION CROSSOVER KOKO
 startCross=clock();
 for(iii=0;iii<individuos;iii+=2)
       {
       R1=1+rand()%(tamanonodo-1);  tempR1=0;tempR2=0;
       R2=1+rand()%(tamanonodo-1);
 for(i=0;i<tamanonodo-1;i++)
       {
  if((R1!=R2)&&(MT[iii][i].nodo==MT[iii+1][R1].nodo))
        { tempR1=i;}
  if((R1!=R2)&&(MT[iii][i].nodo==MT[iii+1][R2].nodo))
        { tempR2=i;}
       }
 //cambia nodo R1
       RAUX[1].nodo=MT[iii][R1].nodo;
       RAUX[1].dem=MT[iii][R1].dem;
       RAUX[1].tiempo=MT[iii][R1].tiempo;
       RAUX[1].vh=MT[iii][R1].vh;
       MT[iii][R1].nodo=MT[iii+1][tempR2].nodo;
       MT[iii][R1].dem=MT[iii+1][tempR2].dem;
       MT[iii][R1].tiempo=MT[iii+1][tempR2].tiempo;
       MT[iii][R1].vh=MT[iii+1][tempR2].vh;
       MT[iii+1][tempR2].nodo=RAUX[1].nodo;
       MT[iii+1][tempR2].dem=RAUX[1].dem;
       MT[iii+1][tempR2].tiempo=RAUX[1].tiempo;
       MT[iii+1][tempR2].vh=RAUX[1].vh;
       //cambia nodo R2
       RAUX[2].nodo=MT[iii][R2].nodo;
       RAUX[2].dem=MT[iii][R2].dem;
       RAUX[2].tiempo=MT[iii][R2].tiempo;
       RAUX[2].vh=MT[iii][R2].vh;
```

```
          MT[iii][R2].nodo=MT[iii+1][tempR1].nodo;
          MT[iii][R2].dem=MT[iii+1][tempR1].dem;
          MT[iii][R2].tiempo=MT[iii+1][tempR1].tiempo;
          MT[iii][R2].vh=MT[iii+1][tempR1].vh;
          MT[iii+1][tempR1].nodo=RAUX[2].nodo;
          MT[iii+1][tempR1].dem=RAUX[2].dem;
          MT[iii+1][tempR1].tiempo=RAUX[2].tiempo;
          MT[iii+1][tempR1].vh=RAUX[2].vh;
  }//fin del crossover koko
  endCross=clock();
  tiempoCalculadoCross = tiempoCalculadoCross+((endCross - startCross) /
(CLOCKS_PER_SEC * 1.0f));
  printf(".");

  ////////////inicia VECINDAD.
  int encontromenor=0,cuentaSI=0,cuentaNO=0,z=0,chunk_size=5;
  startVecino=clock();
 for(iii=0;iii<individuos;iii++)
 { printf("."); encontromenor=0; cuentaSI=0;cuentaNO=0;
   for(z=0;z<iteraciones;z++)
   {   encontromenor=0;  //
      for(x=0;x<vecinos+1;x++)  //genera n vecinos
      {  encontromenor=0;  //
        for(i=0;i<tamanonodo-1;i++)   //copia el individuo original
         {
          MTVEC[x][i].dem=MT[iii][i].dem;
          MTVEC[x][i].nodo=MT[iii][i].nodo;
           MTVEC[x][i].tiempo=MT[iii][i].tiempo;
           MTVEC[x][i].total=MT[iii][i].total;
           MTVEC[x][i].vh=MT[iii][i].vh;
         }
         RV1=1+rand()%(tamanonodo-1);  tempRV1=0;tempRV2=0;  //genera 2 aleatorios
         RV2=1+rand()%(tamanonodo-1);
         for(i=0;i<tamanonodo-1;i++)
         {
           if((RV1!=RV2)&&(MTVEC[x][i].nodo==MTVEC[x][RV1].nodo))
           { tempRV1=i;}
           if((RV1!=RV2)&&(MTVEC[x][i].nodo==MTVEC[x][RV2].nodo))
           { tempRV2=i;}
         }
         ///checa restricciones
         float demand=0.0; int vecindare=0;
         for(i=0;i<tamanonodo-1;i++)
         {
           if(MTVEC[x][i].vh==MTVEC[x][RV1].vh)   //mtvec, mt
            {
              demand=demand+MTVEC[x][i].dem;  //mt
           //verifica si cumple con el tiempo

                  if((instancia.READYTIME[MTVEC[x][i].nodo]>=instancia.READYTIME[MTVE
                  C[x][tempRV2].nodo])&&(instancia.READYTIME[MTVEC[x][i].nodo]<=instan
                  cia.DUEDATE[MTVEC[x][tempRV2].nodo])&&(instancia.DUEDATE[MTVEC[
                  x][i].nodo]>=instancia.DUEDATE[MTVEC[x][tempRV2].nodo]))
```

```
                {
                  vecindare=1; //no vecindare
                }
            }
        } //del for
        demand=demand-MTVEC[x][RV1].dem;   //mt
        demand=demand+MTVEC[x][tempRV2].dem;  //mt
        if(demand>instancia.C)
        {  vecindare=1;  //no vecindare
        }
    demand=0;
    for(i=0;i<tamanonodo-1;i++)
    {
        if(MTVEC[x][i].vh==MTVEC[x][RV2].vh) //MT
        {
        demand=demand+MTVEC[x][i].dem;  //mt
         //verifica si cumple con el tiempo

        if((instancia.READYTIME[MTVEC[x][i].nodo]>=instancia.READYTIME[MTVEC[x][te
        mpRV1].nodo])&&(instancia.READYTIME[MTVEC[x][i].nodo]<=instancia.DUEDATE[
        MTVEC[x][tempRV1].nodo])&&(instancia.DUEDATE[MTVEC[x][i].nodo]>=instancia.
        DUEDATE[MTVEC[x][tempRV1].nodo]))
        {
            vecindare=1; //no vecindare
        }
        }
    } //del for
    demand=demand-MTVEC[x][RV2].dem;
    demand=demand+MTVEC[x][tempRV1].dem;
    if(demand>instancia.C)
    {
            vecindare=1;  //no vecindare
     }
    demand=0;
    //si cumple
    if(vecindare==0)
    {
       for(i=0;i<tamanonodo-1;i++)   //copia el individuo original
        {
          MTVEC[x+1][i].dem=MT[iii][i].dem;
          MTVEC[x+1][i].nodo=MT[iii][i].nodo;
          MTVEC[x+1][i].tiempo=MT[iii][i].tiempo;
          MTVEC[x+1][i].total=MT[iii][i].total;
          MTVEC[x+1][i].vh=MT[iii][i].vh;
           }
    //cambia nodo RV1 y RV2
        RAUX[1].nodo=MTVEC[x+1][RV1].nodo;
        RAUX[1].dem=MTVEC[x+1][RV1].dem;
        RAUX[1].tiempo=MTVEC[x+1][RV1].tiempo;
        RAUX[1].vh=MTVEC[x+1][RV1].vh;
        MTVEC[x+1][RV1].nodo=MTVEC[x+1][tempRV2].nodo;
        MTVEC[x+1][RV1].dem=MTVEC[x+1][tempRV2].dem;
        MTVEC[x+1][RV1].tiempo=MTVEC[x+1][tempRV2].tiempo;
```

```
    MTVEC[x+1][RV1].vh=MTVEC[x+1][tempRV2].vh;
    MTVEC[x+1][tempRV2].nodo=RAUX[1].nodo;
    MTVEC[x+1][tempRV2].dem=RAUX[1].dem;
    MTVEC[x+1][tempRV2].tiempo=RAUX[1].tiempo;
    MTVEC[x+1][tempRV2].vh=RAUX[1].vh;
     }/////vecindare=0
    else
     {x--;}
    } //for N vecinos


//ordena las cadenas de cada vecino
ordenaVEC();
//verifica cual de los n vecinos es el mejor
//primero calcula distancias
double DISTindividuoNuevo=0,mejor=0,RECORRTOTAL=0;
int NN3,NN4,iy;
for(iy=0;iy<vecinos+1;iy++)
{DISTindividuoNuevo=0;RECORRTOTAL=0;
for(x=0;x<tamanonodo-x;x++)
{
    if(x==0)
    {NN3=0;
    NN4=MTVEC[iy][x].nodo;
    RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
    }
    else
    {   if(x==tamanonodo-1)
        {
          NN3=MTVEC[iy][x].nodo;
          NN4=0;
          RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
         }
        else
        {
        if(MTVEC[iy][x-1].vh!=MTVEC[iy][x].vh)
        {
                  NN3=MTVEC[iy][x-1].nodo;
                  NN4=0;
                  RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                 NN3=0;
                  NN4=MTVEC[iy][x].nodo;
                  RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
         }
         else
         {
          if((MTVEC[iy][x-1].vh==MTVEC[iy][x].vh)&&(x!=tamanonodo-1))
          {
                     NN3=MTVEC[iy][x-1].nodo;
                     NN4=MTVEC[iy][x].nodo;
                     RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
           }
           }
```

```
                }
            }
            }  //for x
        MTVEC[iy][0].total=RECORRTOTAL; //asigna el calculo
        } //for iy
        //segundo compara cual es el mejor de los N vecinos
        double MENOR=0;int menor2=0;
         tiempoI=0; tiempoF=0;
        nitera=z+1;
        for(x=0;x<vecinos+1;x++)
        {  tiempoI=((double)clock()/CLOCKS_PER_SEC);
           if(x==0){MENOR=MT[iii][0].total; menor2=x;}
           if((MENOR>MTVEC[x+1][0].total)&&(MTVEC[x+1][0].total>=OPTIMOKNOW))
            {
            MENOR=MTVEC[x+1][0].total;
            menor2=x+1; encontromenor=1; //encontromenor =1 SI
            tiempoF=((double)clock()/CLOCKS_PER_SEC);
            tiempoF=tiempoF-tiempoI;
            MTVEC[x+1][0].nitera=nitera;
            MTVEC[x+1][0].tiempoF=tiempoF;
            MTVEC[x+1][0].tiempoI=tiempoI;
            tiempoI=0; tiempoF=0;
             }
        } //for
        nitera=0;
        if(encontromenor==1)
            {
            //asigna el mejor de los N vecinos a la poblacion
            for(i=0;i<tamanonodo-1;i++)
            {
              MT[iii][i].dem=MTVEC[menor2][i].dem;
              MT[iii][i].nodo=MTVEC[menor2][i].nodo;
              MT[iii][i].tiempo=MTVEC[menor2][i].tiempo;
              MT[iii][0].total=MTVEC[menor2][0].total;
              MT[iii][0].nitera=MTVEC[menor2][0].nitera;
              MT[iii][0].tiempoI=MTVEC[menor2][0].tiempoI;
              MT[iii][0].tiempoF=MTVEC[menor2][0].tiempoF;
              MT[iii][i].vh=MTVEC[menor2][i].vh;
            }
            } //finsi de encontromenor=1
      }//fin del for z iteraciones
      printf("I");
    } //for iii individuos (fase vecindad)

  endVecino=clock();
  tiempoCalculadoVecino = tiempoCalculadoVecino+((endVecino - startVecino) /
(CLOCKS_PER_SEC * 1.0f));
  //imprime resultados en el archivo
   for(i=0;i<individuos;i++)
   {
     if ((MT[i][0].total>=OPTIMOKNOW)&&(MT[i][0].nitera>0))
     {
       fprintf(f,"\n Generacion: %i, individuo: %d, MT con total:%f,
```

```
                    NumItera:%i",ga,i,MT[i][0].total,MT[i][0].nitera);
    }
  }
  printf(".");
}//end del while   GA
  endGA=clock();
  tiempoCalculadopob = (endpob - startpob) / (CLOCKS_PER_SEC * 1.0f);
  tiempoCalculadoGA = (endGA - startGA) / (CLOCKS_PER_SEC * 1.0f);
  printf("@");
  fprintf(f,"\n Tiempo total del Algoritmo Genetico:%f segundos",tiempoCalculadoGA);
  fprintf(f,"\n TiempoCalculadoBest:%f seg",tiempoCalculadoBest);
  fprintf(f,"\n TiempoCalculadoCross:%f seg",tiempoCalculadoCross);
  fprintf(f,"\n TiempoCalculadoVecino:%f seg",tiempoCalculadoVecino);
  fprintf(f,"\n tiempoCalculadoGeneraPoblacionInicial:%f seg",tiempoCalculadopob);
  //calculadisttotal();
}

void genpobinialgo0()
{
   int DisT=0;
   double RECORRTOTAL;
   //calcula individuos
   for(iiiiii=0;iiiiii<individuos;iiiiii++)
   {   RECORRTOTAL=CALCULARUTAS();
       printf("\nINDIVIDUO %d _____________\n(",iiiiii);
       for(jjjjjj=0;jjjjjj<tamanonodo-1;jjjjjj++)    // guarda la matriz de individuos soluciones
                 FACTIBLES  //aqui se puede agregar la lista tabï¿½
       {
            MT[iiiiii][jjjjjj].dem=Resultado[jjjjjj].DemNodo;
            MT[iiiiii][jjjjjj].nodo=Resultado[jjjjjj].Norigen;
            MT[iiiiii][jjjjjj].vh=Resultado[jjjjjj].vh;
            MT[iiiiii][jjjjjj].tiempo=Resultado[jjjjjj].tiempo;
             if (jjjjjj==0)
             {
               MT[iiiiii][0].total=RECORRTOTAL;
             }
            printf("\nnodo %d tiempo %d Veh
                %d",MT[iiiiii][jjjjjj].nodo,MT[iiiiii][jjjjjj].tiempo,MT[iiiiii][jjjjjj].vh);
       }
       fprintf(f,")");
 }
}

double CALCULARUTAS()
{
int i,j,LTS,DELTA,ELE;
char cadena[300];
FILE  *archDatos;
int repetir;
float DemandaT=0.0;
double RecorridoT=0.0,menor=90000,RECORRTOTAL=0;
srand((unsigned)time(NULL));
 //Calcula muchas rutas
```

```
 if (rrr!=1)
{
   for(i=0;i<tamanonodo-1;i++)
  {
            CNN[i]=Resultado[i].Norigen;
   }
   Permutaciones(CNN,tamanonodo);
}
else if(rrr==1)
{
      for(i=0;i<individuos-1;i++) { for(j=0;j<tamanonodo-2;j++) {      Distancias[i][j].marca=0;
      Resultado[j].DemNodo=0; Resultado[j].Norigen=0; Resultado[j].Ndestino=0;
      Resultado[j].tiempo=0; Resultado[j].vh=0; Resultado[j].valor=0; }}
      rutakmeans();   //manda llamar a la instancia ordenada por kmeans
      rrr++;
}
    //calcula tamaï¿½o de la ruta inicial
    int NN3,NN4;
    for(i=0;i<tamanonodo-1;i++)
    {
        if(i==0)
        {NN3=0;
         NN4=Resultado[i].Norigen;
         RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
         }
         else
        {
         if(i==tamanonodo-1)
             {
              NN3=Resultado[i].Norigen;
              NN4=0;
              RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
             }
             else
             {
               if(Resultado[i-1].vh!=Resultado[i].vh)
               {
                NN3=Resultado[i-1].Norigen;
                NN4=0;
                RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                NN3=0;
                NN4=Resultado[i].Norigen;
                RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
               }
               else
               {
                 if((Resultado[i-1].vh==Resultado[i].vh)&&(i!=tamanonodo-1))
                  {
                        NN3=Resultado[i-1].Norigen;
                        NN4=Resultado[i].Norigen;
                        RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                        }
               }
```

```
          }
         }
    }
    return (RECORRTOTAL);

}//FIN FUNCION CALCULA RUTAS *


void Permutaciones(int cad[2000], int ln) {
  int c; // variable auxiliar para intercambio
  int i=0, j=0,num=0; // variables para bucles
  int n1,n2; //valor aleatoreo
  int cad2[2000];
  int LTS,DELTA,ELE;
  srand((unsigned)time(NULL));
  int N=tamanonodo-1,ALEAT,ARREGLOINI[100],
  ARREGLOINDNR[100],k=0,l=tamanonodo-1,ll=1,finlista=tamanonodo-1;
  for(i = 0; i < ln-1; i++)
  {
          Resultado[i].DemNodo=0;
          Resultado[i].Norigen=0;
          Resultado[i].vh=0;
          Resultado[i].tiempo=0;
  }
  for(i = 0; i < ln-1; i++)
  {
             cad2[i]=instancia.CN[i+1];
  }
  if(rrr!=2)
  {
    num=l-1;
    n1=1+(rand()%num+1)-1;
    n2=1+(rand()%num+1)-1;
    // Intercambio de posiciones
    c = cad2[n1];
    cad2[n1]=cad2[n2];
    cad2[n2] = c;
  }
  else
  {  rrr++;
  }
  //guarda en la lista de soluciones generadas aleatoreamente para la poblacion inicial
  for(i = 0; i < ln-1; i++)
  {     if(cad2[i]!=0)
           {
         Resultado[i].DemNodo=instancia.DEMAND[cad2[i]];
            Resultado[i].Norigen=cad2[i];
            Resultado[i].vh=0;
            Resultado[i].tiempo=0;
           }
  } //for
  for(i=0;i<tamanonodo-1;i++)
          {
```

```
                CNN[i]=cad2[i];
        }
    //para asignar el tiempo que le corresponde
    //TIMEWINDOWS();
    for(i=0;i<tamanonodo-1;i++)
                {
                j=instancia.SERVICETIME[i+1];
                TablaVENTANA[i].E=instancia.READYTIME[i+1];
                LTS=TablaVENTANA[i].E+instancia.SERVICETIME[i+1];
                TablaVENTANA[i].LTServicio=LTS;
                DELTA=rand()%j;
                TablaVENTANA[i].DELTAR=DELTA;
                ELE=TablaVENTANA[i].E+DELTA;
                TablaVENTANA[i].ele=ELE;
                Resultado[i].tiempo=TablaVENTANA[i].ele;
                }
        int V=0,nod=0,D=0,J,x,cuenta=0,demanda;
        demanda=instancia.C; V=1; J=0;
        for(i=0;i<tamanonodo-1;i++)
        {
            if(((D+Resultado[i].DemNodo)<demanda))
           {
                Resultado[i].vh=V;
                D=D+Resultado[i].DemNodo;
                J=0;
           }
           else
           {
                 if((D+Resultado[i].DemNodo)>demanda)
                 {
                 V++;
                 D=0;
                 Resultado[i].vh=V;
                 D=D+Resultado[i].DemNodo;
                 }
                 else
                 {
                  D=D+Resultado[i].DemNodo;
                  Resultado[i].vh=V;
                 J=0;
                 }
            }
        }
    //ordena en orden ascendente de vehiculo 1,2,3...10
    for(x=0;x<tamanonodo-2;x++)
        {
                for(j=0;j<tamanonodo-2;j++)
                {
                if(Resultado[j].vh>Resultado[j+1].vh)
                {
                MTAUX[j].dem=Resultado[j].DemNodo;
                MTAUX[j].nodo=Resultado[j].Norigen;
                MTAUX[j].tiempo=Resultado[j].tiempo;
```

```
                    MTAUX[j].vh=Resultado[j].vh;
                    Resultado[j].DemNodo=Resultado[j+1].DemNodo;
                    Resultado[j].Norigen=Resultado[j+1].Norigen;
                    Resultado[j].tiempo=Resultado[j+1].tiempo;
                    Resultado[j].vh=Resultado[j+1].vh;
                    Resultado[j+1].DemNodo=MTAUX[j].dem;
                    Resultado[j+1].Norigen=MTAUX[j].nodo;
                    Resultado[j+1].tiempo=MTAUX[j].tiempo;
                    Resultado[j+1].vh=MTAUX[j].vh;
                    }
                    }
  }
  //ORDENA EN BASE AL TIEMPO
  for(x=0; x<tamanonodo-1;x++)
        {
        for(j=0;j<tamanonodo-1;j++)
        {

        if((Resultado[j].tiempo>Resultado[j+1].tiempo)&&(Resultado[j].vh==Resultado[j+1].v
        h))
        {
          MTAUX[j].dem=Resultado[j].DemNodo;
          MTAUX[j].nodo=Resultado[j].Norigen;
          MTAUX[j].tiempo=Resultado[j].tiempo;
          MTAUX[j].vh=Resultado[j].vh;
          Resultado[j].DemNodo=Resultado[j+1].DemNodo;
          Resultado[j].Norigen=Resultado[j+1].Norigen;
          Resultado[j].tiempo=Resultado[j+1].tiempo;
          Resultado[j].vh=Resultado[j+1].vh;
          Resultado[j+1].DemNodo=MTAUX[j].dem;
          Resultado[j+1].Norigen=MTAUX[j].nodo;
          Resultado[j+1].tiempo=MTAUX[j].tiempo;
          Resultado[j+1].vh=MTAUX[j].vh;
        }
        }
  }
}


void rutakmeans(void)
{
        int i,j,LTS,DELTA,ELE;
        char cadena[300];
        FILE  *archDatos;
        int repetir;
        float DemandaT=0.0;
        double RecorridoT=0.0,menor=90000,RECORRTOTAL=0;
        srand((unsigned)time(NULL));
        printf("Proporcione el Archivo que contiene la poblacion inicial: ");
        scanf("%s", archivo); rrr++;
        int cuentador=0; char c;

        if((archDatos=fopen(archivo,"rt"))==NULL)
```

```
 { printf(" No existe el archivo de datos ==> [%s]%c",archivo,7);
 }
 printf("\n            El archivo es %s", archivo);
 fgets(cadena,300,archDatos);
 for(repetir=0;repetir<tamanonodo-1;repetir++)
 { //fgets(cadena,300,archDatos);
 fscanf(archDatos,"%d\t",&Resultado[repetir].Norigen);
 fscanf(archDatos,"%d\t",&Resultado[repetir].vh);
 Resultado[repetir].DemNodo=instancia.DEMAND[Resultado[repetir].Norigen];
 }

for(i=0;i<tamanonodo-1;i++)
 {
 j=instancia.SERVICETIME[i+1]; //Bench[1].Service;
 TablaVENTANA[i].E=instancia.READYTIME[i+1]; //Bench[i].ReadyT;
 LTS=TablaVENTANA[i].E+instancia.SERVICETIME[i+1]; // Bench[i].Service;
 TablaVENTANA[i].LTServicio=LTS;
 DELTA=rand()%j;
 TablaVENTANA[i].DELTAR=DELTA;
 ELE=TablaVENTANA[i].E+DELTA;
 TablaVENTANA[i].ele=ELE;
 Resultado[i].tiempo=TablaVENTANA[i].ele;
 }
 int V=0,nod=0,D=0,J,x,cuenta=0,demanda;
 demanda=instancia.C; V=1; J=0;
 for(i=0;i<tamanonodo-1;i++)
 {
     if(((D+Resultado[i].DemNodo)<demanda))
             {
             //Resultado[i].vh=V;
             D=D+Resultado[i].DemNodo;
            J=0;
             }
             else
             {
             if((D+Resultado[i].DemNodo)>demanda)
             {
             V++;
             D=0;
             Resultado[i].vh=V;
             D=D+Resultado[i].DemNodo;
             }
             }
 }
//ordena en orden ascendente de vehiculo 1,2,3...10
for(x=0;x<tamanonodo-2;x++)
 {
             for(j=0;j<tamanonodo-2;j++)
             {
             if(Resultado[j].vh>Resultado[j+1].vh)
             {
             MTAUX[j].dem=Resultado[j].DemNodo;
            MTAUX[j].nodo=Resultado[j].Norigen;
```

```
                    MTAUX[j].tiempo=Resultado[j].tiempo;
                    MTAUX[j].vh=Resultado[j].vh;
                   Resultado[j].DemNodo=Resultado[j+1].DemNodo;
                   Resultado[j].Norigen=Resultado[j+1].Norigen;
                    Resultado[j].tiempo=Resultado[j+1].tiempo;
                   Resultado[j].vh=Resultado[j+1].vh;
                   Resultado[j+1].DemNodo=MTAUX[j].dem;
                    Resultado[j+1].Norigen=MTAUX[j].nodo;
                    Resultado[j+1].tiempo=MTAUX[j].tiempo;
                    Resultado[j+1].vh=MTAUX[j].vh;
                    }
                    }
   }
   //ORDENA EN BASE AL TIEMPO
   for(x=0; x<tamanonodo-1;x++)
          {
          for(j=0;j<tamanonodo-1;j++)
          {

          if((Resultado[j].tiempo>Resultado[j+1].tiempo)&&(Resultado[j].vh==Resultado[j+1].v
          h))
           {
             MTAUX[j].dem=Resultado[j].DemNodo;
             MTAUX[j].nodo=Resultado[j].Norigen;
             MTAUX[j].tiempo=Resultado[j].tiempo;
             MTAUX[j].vh=Resultado[j].vh;
             Resultado[j].DemNodo=Resultado[j+1].DemNodo;
             Resultado[j].Norigen=Resultado[j+1].Norigen;
             Resultado[j].tiempo=Resultado[j+1].tiempo;
             Resultado[j].vh=Resultado[j+1].vh;
             Resultado[j+1].DemNodo=MTAUX[j].dem;
             Resultado[j+1].Norigen=MTAUX[j].nodo;
             Resultado[j+1].tiempo=MTAUX[j].tiempo;
             Resultado[j+1].vh=MTAUX[j].vh;
             }
          }
     }

}


void ordenaVEC()
{

 int V=0,i,x,j,repetir;
  for(i=0;i<vecinos+1;i++)
  {
    //asigna temporalmente el ready time como tiempo
   for(repetir=0;repetir<tamanonodo-1;repetir++)
    {  //asigna temporalmente el ready time como tiempo
       MTVEC[i][repetir].tiempo=instancia.READYTIME[Resultado[repetir].Norigen];
     }
   //calcula demandas
```

```
int nod=0,D=0,J,cuenta=0,demanda;
demanda=instancia.C; V=1; J=0;
for(x=0;x<tamanonodo-1;x++)
{
 if(((D+MTVEC[i][x].dem)<demanda))
  {
                  MTVEC[i][x].vh=V;
                  D=D+MTVEC[i][x].dem;
                 J=0;
   }
 else
  {
                              if((D+MTVEC[i][x].dem)>demanda)
                              {
                                V++;
                                D=0;
                                MTVEC[i][x].vh=V;
                                D=D+MTVEC[i][x].dem;
                              }
                              else
                              {
                               D=D+MTVEC[i][x].dem;
                               MTVEC[i][x].vh=V;
                              J=0;
                               }
        }
 } //fin for x

 //ordena en orden ascendente de vehiculo 1,2,3...10
 for(x=0;x<tamanonodo-2;x++)
 {
                  for(j=0;j<tamanonodo-2;j++)
                  {
                  if(MTVEC[i][j].vh>MTVEC[i][j+1].vh)
                  {
                  MTAUX[j].dem=MTVEC[i][j].dem;
                 MTAUX[j].nodo=MTVEC[i][j].nodo;
                  MTAUX[j].tiempo=MTVEC[i][j].tiempo;
                  MTAUX[j].vh=MTVEC[i][j].vh;
                  MTVEC[i][j].dem=MTVEC[i][j+1].dem;
                 MTVEC[i][j].nodo=MTVEC[i][j+1].nodo;
                  MTVEC[i][j].tiempo=MTVEC[i][j+1].tiempo;
                 MTVEC[i][j].vh=MTVEC[i][j+1].vh;
                 MTVEC[i][j+1].dem=MTAUX[j].dem;
                  MTVEC[i][j+1].nodo=MTAUX[j].nodo;
                  MTVEC[i][j+1].tiempo=MTAUX[j].tiempo;
                  MTVEC[i][j+1].vh=MTAUX[j].vh;
                  }
                  }
}

////////////////////////////////////////////////////////////////////////////
//para asignar el tiempo que le corresponde
```

```
////////////////////////////////////////////////////////////////////////////
 int R=0,R2=0,RFIN=9999,ok=0;
 for(j=0;j<tamanonodo-1;j++)
        {
         if(MTVEC[i][j].vh==MTVEC[i][j+1].vh)
        {
        R=R+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
        for(x=j;x<tamanonodo-1;x++)
        {
          if(MTVEC[i][x].tiempo!=0)
          {   if((R<MTVEC[i][x].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][x].vh))//&&(R<RFIN))
                    {
                     MTVEC[i][j].tiempo=R;
                      RFIN=MTVEC[i][x].tiempo;
                      ok=x;
                      break;
                    }
                   else
                    {

                  if((R>MTVEC[i][x].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][x].vh))//&&(R>RFIN
                  ))
                   {
                        R=instancia.DUEDATE[MT[i][x].nodo];
                       RFIN=R;
                       //mueve el mayor tiempo a donde le corresponde
                       MTAUX[x].dem=MTVEC[i][x].dem;
                       MTAUX[x].nodo=MTVEC[i][x].nodo;
                        MTAUX[x].tiempo=MTVEC[i][x].tiempo;
                        MTAUX[x].vh=MTVEC[i][x].vh;
                       MTVEC[i][x].dem=MTVEC[i][j].dem;
                       MTVEC[i][x].nodo=MTVEC[i][j].nodo;
                       MTVEC[i][x].tiempo=MTVEC[i][j].tiempo;
                       MTVEC[i][x].vh=MTVEC[i][j].vh;
                       MTVEC[i][j].dem=MTAUX[x].dem;
                        MTVEC[i][j].nodo=MTAUX[x].nodo;
                        MTVEC[i][j].tiempo=MTAUX[x].tiempo;
                        MTVEC[i][j].vh=MTAUX[x].vh;
                        ok=999;
                        break;
                              }
                    }//fin else
                    } //fin si !=0
                   }//fin for
                  }
        else if(MTVEC[i][j].vh!=MTVEC[i][j+1].vh)
        {
         R=R+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
         R=0; RFIN=9999;
        }
       }//fin for

 for(j=0;j<tamanonodo-1;j++)
```

```
    {
     if((MTVEC[i][j].tiempo==0)&&(MTVEC[i][j].vh==MTVEC[i][j+1].vh))
     {
         R=MTVEC[i][j-1].tiempo+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
         MTVEC[i][j].tiempo=R;
      }

       if((MTVEC[i][j].vh!=MTVEC[i][j+1].vh)&&(MTVEC[i][j].tiempo==0))
             {
              R=MTVEC[i][j-1].tiempo+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
             MTVEC[i][j].tiempo=R;
              }
       }//fin for


  for(x=0; x<tamanonodo-1;x++)
  {
    for(j=0;j<tamanonodo-1;j++)
    {
      if((MT[i][j].tiempo>MTVEC[i][j+1].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][j+1].vh))
             {
             MTAUX[j].dem=MTVEC[i][j].dem;
             MTAUX[j].nodo=MTVEC[i][j].nodo;
             MTAUX[j].tiempo=MTVEC[i][j].tiempo;
             MTAUX[j].vh=MTVEC[i][j].vh;
             MTVEC[i][j].dem=MTVEC[i][j+1].dem;
             MTVEC[i][j].nodo=MTVEC[i][j+1].nodo;
             MTVEC[i][j].tiempo=MTVEC[i][j+1].tiempo;
             MTVEC[i][j].vh=MTVEC[i][j+1].vh;
             MTVEC[i][j+1].dem=MTAUX[j].dem;
             MTVEC[i][j+1].nodo=MTAUX[j].nodo;
             MTVEC[i][j+1].tiempo=MTAUX[j].tiempo;
             MTVEC[i][j+1].vh=MTAUX[j].vh;
             }
          }
  }
  }//fin del i
}
```

APÉNDICE E

ALGORITMO PARALELO CÓDIGO DEL ALGORITMO GA-VRPTW-PN

```
#include <omp.h>
#include <stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<string.h>
#include<malloc.h>
#include <ctype.h>
#include <values.h>
#define N2 10
#define CHUNKSIZE 4
char archivo[50];
int tamanonodo=0,nodo;
FILE *f = NULL;//filefile
char name[5];
int CUENTA,iiiiii=0,jjjjjj=0,rrr=1;
int INDIVIDUO;
int NUMNOD=101,CAPACIDADVEHI=200;
float
tiempoCalculadoGA,tiempoCalculadoCross,tiempoCalculadoVecino,tiempoCalculado,tiempo
CalculadoBest,tiempoCalculadopob;
double Tinicio=0,Tfin,tiempoI,tiempoF;
double mascortaPARNODO(int I,int F);
int casosCSP(int NumParejas);
int calculaNODOINCONSISTENTE();
void algoritmogeneticoSIMPLE();
double MASCORTA();
double VENTANAS();
void TIMEWINDOWS();
void divideinstancia();
void algoritmogeneticoBPP();
void algoritmogeneticomtsp();
void MATDistancias();
void genpobinialgo0();
void calculadisttotal();
void algoritmogeneticoKOKO();
int combinaPARES();
double CALCULARUTAS();
void TIMEWINDOWS();
void rutakmeans(void);
void Permutaciones(int cad[2000], int I);
int CNN[101];
float OPTIMOKNOW;
int individuos=0,generaciones=0,vecinos=0,iteraciones=0;
void ordenaVEC();
int omp_get_num_threads(void);
int omp_get_max_threads(void);
```

```
int omp_get_thread_num(void);
int omp_get_num_procs(void);
void omp_set_dynamic(int dynamic_threads);
int omp_get_dynamic(void);
int omp_in_parallel(void);
void omp_set_nested(int nested);
int omp_get_nested(void);


struct  datosinstancia{
          char instancia[5];
           int VN;
           int       C;
           int       CN[101];
           int       XCOOR[101];
           int       YCOOR[101];
           int       DEMAND[101];
           int       READYTIME[101];
           int       DUEDATE[101];
           int       SERVICETIME[101];
 }instancia;

//estructura que guarda resultados
struct  Resultados{
   int Norigen;
   int Ndestino;
   double valor;
   float DemNodo;
   int vh;
   int tiempo;
}Resultado[1001];

//estructura que guarda las distancias
struct  datos{
   double dato;
   int marca;
}Distancias[101][101],Original[101][101],OriginalFinal[101][101],DistanciaNueva[101][101],T
empoDistancias[101][101];

struct QQ{
           int nodo;
           int vh;
           int dem;
           int tiempo;
}R[1001], RAUX[50];

struct QA0
{
           int nodo;
           int vh;
           int dem;
           int tiempo;
           double total;
```

```
        double tiempoI;
        double tiempoF;
        int nitera;
}MT[2001][1000];

struct QA1
{
        int nodo;
        int vh;
        int dem;
        int tiempo;
        double total;
        double tiempoI;
        double tiempoF;
        int nitera;
}MTAUX[2001];

struct QA2
{
        int nodo;
        int vh;
        int dem;
        int tiempo;
        double total;
        double tiempoI;
        double tiempoF;
        int nitera;
}MTVEC[10000][100];

struct VENTANAS{
   int Nodo;
   int E;
   int LTServicio;
   int DELTAR;
   int ele;
}TablaVENTANA[1001];


int main(int argc,char *argv[])
{       float a[N2], b[N2], total[N2]; int i;
        int cuenta=0,NumCasos=1,ZeroNodo=0,bandera=1;
        float DemandaT=0;
        double DISTindividuoNuevo=0;
        int factible=0,nofactible=0,ham=0;
        int BANDERAPOB=0;
        char cadena[300];
        cadena[0]='\0'; FILE  *archDatos;
        clock_t start, end;
        int repetir;
        srand((unsigned)time(NULL));
        sprintf(name,"resultados-GAx.txt");
        if(!(f=fopen(name,"w"))){printf("error al abrir el archivo");return 0;}
        printf("Proporcione el Archivo de la instancia a resolver: ");
```

```
    scanf("%s", archivo);
    if((archDatos=fopen(archivo,"rt"))==NULL)
    { printf(" No existe el archivo de datos ==> [%s]%c",archivo,7);}
    printf("\n    El archivo es %s", archivo);
    fprintf(f,"\n El archivo es %s", archivo);
    fgets(instancia.instancia,5,archDatos);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
    fscanf(archDatos,"%d\t",&instancia.VN);
    fscanf(archDatos,"%d\t",&instancia.C);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
    fgets(cadena,300,archDatos);
    for(repetir=0;repetir<101;repetir++)   //41
        {
          fscanf(archDatos,"%d\t",&instancia.CN[repetir]);
          fscanf(archDatos,"%d\t",&instancia.XCOOR[repetir]);
          fscanf(archDatos,"%d\t",&instancia.YCOOR[repetir]);
          fscanf(archDatos,"%d\t",&instancia.DEMAND[repetir]);
          fscanf(archDatos,"%d\t",&instancia.READYTIME[repetir]);
          fscanf(archDatos,"%d\t",&instancia.DUEDATE[repetir]);
          fscanf(archDatos,"%d\t",&instancia.SERVICETIME[repetir]);
        }
    tamanonodo=repetir;
    MATDistancias();
    algoritmogeneticoKOKO();
    fclose(archDatos);
    fclose(f);
    printf("\n\n\tFin.");
    return(0);
}


void MATDistancias()
{
   float a,b,ra,rb,rc; int k=0,j=0,i=0;
   for (k=0;k<tamanonodo-2;k++)     /*LLENADO DE MATRIZ CON CEROS*/
   { for (j=0;j<tamanonodo-2;j++) {OriginalFinal[k][j].dato=0;}
   }
   for (k=0;k<tamanonodo-1;k++)  /*CALCULA DISTANCIAS y llena matriz*/
   {
    for (j=0;j<tamanonodo-1;j++)
    {
          if(k==j)
          {
            OriginalFinal[k][j].dato=0.0;
          }
          else
          {  if (OriginalFinal[k][j].dato==0)   //Distancias[i][j].dato==0)
            {
              a=instancia.XCOOR[j]- instancia.XCOOR[k];
```

```
            b=instancia.YCOOR[j]-instancia.YCOOR[k];
            OriginalFinal[k][j].dato=sqrt((a*a)+(b*b));
          }
        }
    }
  }
}

void algoritmogeneticoKOKO()
{ int ga=0,i=0,j=0,x=0,iii=0,R1,R2,tempR1,tempR2,RV1,RV2,tempRV1,tempRV2,nitera=0;
  double DisT=0.0;
  clock_t startGA, endGA,startBest,endBest,startCross,endCross,startVecino,endVecino;
  clock_t startpob,endpob;
  printf("\nEmpezo el algoritmo genetico simple propuesto por koko");
  printf("\n(Seleccion=Best, Crossover=koko, Muta=VECINDAD");
  printf("\nPorfavor introduzca el optimo de la instancia(827.3): ");
  scanf("%g",&OPTIMOKNOW);
  printf("\nPorfavor introduzca el numero de individuos(1000): ");
  scanf("%i",&individuos);
  printf("\nPorfavor introduzca el numero de generaciones (20): ");
  scanf("%i",&generaciones);
  printf("\nPorfavor introduzca el numero de vecinos(10): ");
  scanf("%i",&vecinos);
  printf("\nPorfavor introduzca el numero de iteraciones(9900): ");
  scanf("%i",&iteraciones);
  startpob=clock();
  genpobinialgo0();
  endpob=clock();
  startGA=clock();

  while(ga!=generaciones)  //criterio de paro
  { printf("."); ga++;
   printf("\nGeneracion: %i",ga);
  //////SECCION BEST
  startBest=clock();
  for(i=0;i<individuos;i++)
          {
          for(j=0;j<individuos;j++)
          {
          if(MT[i][0].total< MT[j][0].total)
          {
             for(x=0;x<tamanonodo-1;x++)
                 {
                     MTAUX[x].dem=MT[i][x].dem;
                    MTAUX[x].nodo=MT[i][x].nodo;
                     MTAUX[x].tiempo=MT[i][x].tiempo;
                     MTAUX[x].vh=MT[i][x].vh;
                     if (x==0)
                     {
                     MTAUX[0].total=MT[i][0].total;
                     }
                    MT[i][x].dem=MT[j][x].dem;
                    MT[i][x].nodo=MT[j][x].nodo;
```

```
                MT[i][x].tiempo=MT[j][x].tiempo;
                MT[i][x].vh=MT[j][x].vh;
                  if (x==0)
                  {
                   MT[i][0].total=MT[j][0].total;
                  }
                 MT[j][x].dem=MTAUX[x].dem;
                  MT[j][x].nodo=MTAUX[x].nodo;
                  MT[j][x].tiempo=MTAUX[x].tiempo;
                  MT[j][x].vh=MTAUX[x].vh;
                  if (x==0)
                  {
                  MT[j][0].total=MTAUX[0].total;
                  MTAUX[x].total=0;
                  }
              }
            }
           }
 }
endBest=clock();
tiempoCalculadoBest = tiempoCalculadoBest +((endBest - startBest) /
              (CLOCKS_PER_SEC * 1.0f));
printf(".");
////SECCION CROSSOVER KOKO
startCross=clock();
for(iii=0;iii<individuos;iii+=2)
       {
       int crossrate=1;
       R1=1+rand()%(tamanonodo-1);  tempR1=0;tempR2=0;
       R2=1+rand()%(tamanonodo-1);
       for(i=0;i<tamanonodo-1;i++)
       {
      if((R1!=R2)&&(MT[iii][i].nodo==MT[iii+1][R1].nodo))
        { tempR1=i;}
      if((R1!=R2)&&(MT[iii][i].nodo==MT[iii+1][R2].nodo))
        { tempR2=i;}
       }
  //cambia nodo R1
RAUX[1].nodo=MT[iii][R1].nodo;
RAUX[1].dem=MT[iii][R1].dem;
RAUX[1].tiempo=MT[iii][R1].tiempo;
RAUX[1].vh=MT[iii][R1].vh;
MT[iii][R1].nodo=MT[iii+1][tempR2].nodo;
MT[iii][R1].dem=MT[iii+1][tempR2].dem;
MT[iii][R1].tiempo=MT[iii+1][tempR2].tiempo;
MT[iii][R1].vh=MT[iii+1][tempR2].vh;
MT[iii+1][tempR2].nodo=RAUX[1].nodo;
MT[iii+1][tempR2].dem=RAUX[1].dem;
MT[iii+1][tempR2].tiempo=RAUX[1].tiempo;
MT[iii+1][tempR2].vh=RAUX[1].vh;
//cambia nodo R2
RAUX[2].nodo=MT[iii][R2].nodo;
RAUX[2].dem=MT[iii][R2].dem;
```

```
  RAUX[2].tiempo=MT[iii][R2].tiempo;
  RAUX[2].vh=MT[iii][R2].vh;
  MT[iii][R2].nodo=MT[iii+1][tempR1].nodo;
  MT[iii][R2].dem=MT[iii+1][tempR1].dem;
  MT[iii][R2].tiempo=MT[iii+1][tempR1].tiempo;
  MT[iii][R2].vh=MT[iii+1][tempR1].vh;
  MT[iii+1][tempR1].nodo=RAUX[2].nodo;
  MT[iii+1][tempR1].dem=RAUX[2].dem;
  MT[iii+1][tempR1].tiempo=RAUX[2].tiempo;
  MT[iii+1][tempR1].vh=RAUX[2].vh;
 }//fin del crossover koko
 endCross=clock();
 tiempoCalculadoCross = tiempoCalculadoCross+((endCross - startCross) /
            (CLOCKS_PER_SEC * 1.0f));
 printf(".");

 ////////////inicia VECINDAD.
#pragma omp parallel shared(MTVEC,MT) private(iii)
{
 int encontromenor=0,cuentaSI=0,cuentaNO=0,z=0,chunk_size;
 startVecino=clock();chunk_size=CHUNKSIZE;
 #pragma omp for schedule(dynamic,chunk_size)
  for(iii=0;iii<individuos;iii++)
  { printf("."); encontromenor=0; cuentaSI=0;cuentaNO=0;
   for(z=0;z<iteraciones;z++)
   {   encontromenor=0;  //
      for(x=0;x<vecinos+1;x++)  //genera n vecinos
         {  encontromenor=0;  //
         for(i=0;i<tamanonodo-1;i++)   //copia el individuo original
         {
           MTVEC[x][i].dem=MT[iii][i].dem;
           MTVEC[x][i].nodo=MT[iii][i].nodo;
           MTVEC[x][i].tiempo=MT[iii][i].tiempo;
           MTVEC[x][i].total=MT[iii][i].total;
           MTVEC[x][i].vh=MT[iii][i].vh;
         }
         RV1=1+rand()%(tamanonodo-1);  tempRV1=0;tempRV2=0;  //genera 2 aleatorios
         RV2=1+rand()%(tamanonodo-1);
         for(i=0;i<tamanonodo-1;i++)
         {
         if((RV1!=RV2)&&(MTVEC[x][i].nodo==MTVEC[x][RV1].nodo))
         { tempRV1=i;}
         if((RV1!=RV2)&&(MTVEC[x][i].nodo==MTVEC[x][RV2].nodo))
         { tempRV2=i;}
         }
         //checa restricciones
         float demand=0.0; int vecindare=0;
         for(i=0;i<tamanonodo-1;i++)
         {
         if(MTVEC[x][i].vh==MTVEC[x][RV1].vh)   //mtvec, mt
           {
           demand=demand+MTVEC[x][i].dem;  //mt
           //verifica si cumple con el tiempo
```

```
if((instancia.READYTIME[MTVEC[x][i].nodo]>=instancia.READYTIME[MTVEC[x][te
mpRV2].nodo])&&(instancia.READYTIME[MTVEC[x][i].nodo]<=instancia.DUEDATE[
MTVEC[x][tempRV2].nodo])&&(instancia.DUEDATE[MTVEC[x][i].nodo]>=instancia.
DUEDATE[MTVEC[x][tempRV2].nodo]))
 {
vecindare=1; //no vecindare
 }
   }
} //del for
demand=demand-MTVEC[x][RV1].dem;   //mt
demand=demand+MTVEC[x][tempRV2].dem;  //mt
if(demand>instancia.C)
{  vecindare=1;  //no vecindare
 }
demand=0;
for(i=0;i<tamanonodo-1;i++)
 {
  if(MTVEC[x][i].vh==MTVEC[x][RV2].vh) //MT
  {
   demand=demand+MTVEC[x][i].dem;  //mt
   //verifica si cumple con el tiempo

          if((instancia.READYTIME[MTVEC[x][i].nodo]>=instancia.READYTIME[MTVE
          C[x][tempRV1].nodo])&&(instancia.READYTIME[MTVEC[x][i].nodo]<=instan
          cia.DUEDATE[MTVEC[x][tempRV1].nodo])&&(instancia.DUEDATE[MTVEC[
          x][i].nodo]>=instancia.DUEDATE[MTVEC[x][tempRV1].nodo]))
          {
             vecindare=1; //no vecindare
          }
  }
 } //del for
demand=demand-MTVEC[x][RV2].dem;
demand=demand+MTVEC[x][tempRV1].dem;
 if(demand>instancia.C)
 {
    vecindare=1;  //no vecindare
 }
demand=0;
//si cumple
 if(vecindare==0)
 {
  for(i=0;i<tamanonodo-1;i++)   //copia el individuo original
  {
  MTVEC[x+1][i].dem=MT[iii][i].dem;
  MTVEC[x+1][i].nodo=MT[iii][i].nodo;
  MTVEC[x+1][i].tiempo=MT[iii][i].tiempo;
  MTVEC[x+1][i].total=MT[iii][i].total;
  MTVEC[x+1][i].vh=MT[iii][i].vh;
   }
//cambia nodo RV1 y RV2
RAUX[1].nodo=MTVEC[x+1][RV1].nodo;
RAUX[1].dem=MTVEC[x+1][RV1].dem;
```

```
RAUX[1].tiempo=MTVEC[x+1][RV1].tiempo;
RAUX[1].vh=MTVEC[x+1][RV1].vh;
MTVEC[x+1][RV1].nodo=MTVEC[x+1][tempRV2].nodo;
MTVEC[x+1][RV1].dem=MTVEC[x+1][tempRV2].dem;
MTVEC[x+1][RV1].tiempo=MTVEC[x+1][tempRV2].tiempo;
MTVEC[x+1][RV1].vh=MTVEC[x+1][tempRV2].vh;
MTVEC[x+1][tempRV2].nodo=RAUX[1].nodo;
MTVEC[x+1][tempRV2].dem=RAUX[1].dem;
MTVEC[x+1][tempRV2].tiempo=RAUX[1].tiempo;
MTVEC[x+1][tempRV2].vh=RAUX[1].vh;
 }/////vecindare=0
else
 {x--;}
} //for N vecinos

//ordena las cadenas de cada vecino
ordenaVEC();
//verifica cual de los n vecinos es el mejor
//primero calcula distancias
double DISTindividuoNuevo=0,mejor=0,RECORRTOTAL=0;
int NN3,NN4,iy;
for(iy=0;iy<vecinos+1;iy++)
{DISTindividuoNuevo=0;RECORRTOTAL=0;
for(x=0;x<tamanonodo-x;x++)
{
if(x==0)
{NN3=0;
NN4=MTVEC[iy][x].nodo;
RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
}
else
{    if(x==tamanonodo-1)
            {
            NN3=MTVEC[iy][x].nodo;
            NN4=0;
            RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
            }
            else
            {
            if(MTVEC[iy][x-1].vh!=MTVEC[iy][x].vh)
              {
                NN3=MTVEC[iy][x-1].nodo;
                NN4=0;
                RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                NN3=0;
                NN4=MTVEC[iy][x].nodo;
                RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
              }
              else
              {
              if((MTVEC[iy][x-1].vh==MTVEC[iy][x].vh)&&(x!=tamanonodo-1))
                      {
                       NN3=MTVEC[iy][x-1].nodo;
```

```
                                NN4=MTVEC[iy][x].nodo;
                                RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                               }
                          }
                        }
                    }
                    }  //for x
          MTVEC[iy][0].total=RECORRTOTAL; //asigna el calculo
          } //for iy
          //segundo compara cual es el mejor de los N vecinos
          double MENOR=0;int menor2=0;
          tiempoI=0; tiempoF=0;
          nitera=z+1;
          for(x=0;x<vecinos+1;x++)
              {  tiempoI=((double)clock()/CLOCKS_PER_SEC);
                 if(x==0){MENOR=MT[iii][0].total; menor2=x;}
                 if((MENOR>MTVEC[x+1][0].total)&&(MTVEC[x+1][0].total>=OPTIMOKNOW))
                  {
                  MENOR=MTVEC[x+1][0].total;
                  menor2=x+1; encontromenor=1; //encontromenor =1 SI
                  tiempoF=((double)clock()/CLOCKS_PER_SEC);
                  tiempoF=tiempoF-tiempoI;
                  MTVEC[x+1][0].nitera=nitera;
                  MTVEC[x+1][0].tiempoF=tiempoF;
                  MTVEC[x+1][0].tiempoI=tiempoI;
                  tiempoI=0; tiempoF=0;
                 }
              } //for
          nitera=0;
          if(encontromenor==1)
              {
              //asigna el mejor de los N vecinos a la poblacion
              for(i=0;i<tamanonodo-1;i++)
              {
                MT[iii][i].dem=MTVEC[menor2][i].dem;
                MT[iii][i].nodo=MTVEC[menor2][i].nodo;
                MT[iii][i].tiempo=MTVEC[menor2][i].tiempo;
                MT[iii][0].total=MTVEC[menor2][0].total;
                MT[iii][0].nitera=MTVEC[menor2][0].nitera;
                MT[iii][0].tiempoI=MTVEC[menor2][0].tiempoI;
                MT[iii][0].tiempoF=MTVEC[menor2][0].tiempoF;
                MT[iii][i].vh=MTVEC[menor2][i].vh;
              }
              } //finsi de encontromenor=1

       }//fin del for z iteraciones
       printf("I");
     } //for iii individuos (fase vecindad)
    }  //fin del pragama omp parallel shared
    endVecino=clock();
    tiempoCalculadoVecino = tiempoCalculadoVecino+((endVecino - startVecino) /
(CLOCKS_PER_SEC * 1.0f));
```

```
 //imprime resultados en el archivo
   for(i=0;i<individuos;i++)
  {
    if ((MT[i][0].total>=OPTIMOKNOW)&&(MT[i][0].nitera>0))
             {
            fprintf(f,"\n Generacion: %i, individuo: %d, MT con total:%f,
            NumItera:%i",ga,i,MT[i][0].total,MT[i][0].nitera);
             }
  }
  printf(".");
}//end del while   GA
  endGA=clock();
  tiempoCalculadopob = (endpob - startpob) / (CLOCKS_PER_SEC * 1.0f);
  tiempoCalculadoGA = (endGA - startGA) / (CLOCKS_PER_SEC * 1.0f);
  printf("@");
  fprintf(f,"\n Tiempo total del Algoritmo Genetico:%f segundos",tiempoCalculadoGA);
  fprintf(f,"\n TiempoCalculadoBest:%f seg",tiempoCalculadoBest);
  fprintf(f,"\n TiempoCalculadoCross:%f seg",tiempoCalculadoCross);
  fprintf(f,"\n TiempoCalculadoVecino:%f seg",tiempoCalculadoVecino);
  fprintf(f,"\n tiempoCalculadoGeneraPoblacionInicial:%f seg",tiempoCalculadopob);
  //calculadisttotal();
}


void genpobinialgo0()
{

   int DisT=0;
   double RECORRTOTAL;
   //calcula individuos
   for(iiiiii=0;iiiiii<individuos;iiiiii++)
            {   RECORRTOTAL=CALCULARUTAS();
            printf("\nINDIVIDUO %d _____________\n(",iiiiii);
            for(jjjjjj=0;jjjjjj<tamanonodo-1;jjjjjj++)    // guarda la matriz de individuos soluciones
                     FACTIBLES  //aqui se puede agregar la lista tabï¿½
            {
            MT[iiiiii][jjjjjj].dem=Resultado[jjjjjj].DemNodo;
            MT[iiiiii][jjjjjj].nodo=Resultado[jjjjjj].Norigen;
            MT[iiiiii][jjjjjj].vh=Resultado[jjjjjj].vh;
            MT[iiiiii][jjjjjj].tiempo=Resultado[jjjjjj].tiempo;
             if (jjjjjj==0)
             {
               MT[iiiiii][0].total=RECORRTOTAL;
             }
            printf("\nnodo %d tiempo %d Veh
            %d",MT[iiiiii][jjjjjj].nodo,MT[iiiiii][jjjjjj].tiempo,MT[iiiiii][jjjjjj].vh);
             }
            fprintf(f,")");
            }

}

double CALCULARUTAS()
```

```
{
        int i,j,LTS,DELTA,ELE;
        char cadena[300];
        FILE  *archDatos;
        int repetir;
        float DemandaT=0.0;
        double RecorridoT=0.0,menor=90000,RECORRTOTAL=0;
        srand((unsigned)time(NULL));
        //Calcula muchas rutas
       if (rrr!=1)
        {
                for(i=0;i<tamanonodo-1;i++)
                {
                        CNN[i]=Resultado[i].Norigen;
                }
                Permutaciones(CNN,tamanonodo);
        }
        else if(rrr==1)
        {
        for(i=0;i<individuos-1;i++) { for(j=0;j<tamanonodo-2;j++) {Distancias[i][j].marca=0;
Resultado[j].DemNodo=0; Resultado[j].Norigen=0;
        Resultado[j].Ndestino=0; Resultado[j].tiempo=0; Resultado[j].vh=0;
        Resultado[j].valor=0; }}
        rutakmeans();   //manda llamar a la instancia ordenada por kmeans
        rrr++;

        }
        int NN3,NN4;
       for(i=0;i<tamanonodo-1;i++)
        {
        if(i==0)
        {NN3=0;
         NN4=Resultado[i].Norigen;
         RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
        }
        else
        {
        if(i==tamanonodo-1)
         {
          NN3=Resultado[i].Norigen;
          NN4=0;
          RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
         }
         else
         {
         if(Resultado[i-1].vh!=Resultado[i].vh)
           {
            NN3=Resultado[i-1].Norigen;
            NN4=0;
            RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
            NN3=0;
            NN4=Resultado[i].Norigen;
            RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
```

```
            }
            else
             {
              if((Resultado[i-1].vh==Resultado[i].vh)&&(i!=tamanonodo-1))
                {
                      NN3=Resultado[i-1].Norigen;
                      NN4=Resultado[i].Norigen;
                      RECORRTOTAL=RECORRTOTAL+OriginalFinal[NN3][NN4].dato;
                      }
             }
            }
           }
    }
    return (RECORRTOTAL);
}//FIN FUNCION CALCULA RUTAS *


void Permutaciones(int cad[2000], int ln) {
   int c; // variable auxiliar para intercambio
   int i=0, j=0,num=0; // variables para bucles
   int n1,n2; //valor aleatoreo
   int cad2[2000];
   int LTS,DELTA,ELE;
   srand((unsigned)time(NULL));
   int N=tamanonodo-1,ALEAT,ARREGLOINI[100],
  ARREGLOINDNR[100],k=0,l=tamanonodo-1,ll=1,finlista=tamanonodo-1;
 for(i = 0; i < ln-1; i++)
   {
           Resultado[i].DemNodo=0;
           Resultado[i].Norigen=0;
           Resultado[i].vh=0;
           Resultado[i].tiempo=0;
   }
   for(i = 0; i < ln-1; i++)
   {
              cad2[i]=instancia.CN[i+1];
   }
   if(rrr!=2)
   {
     num=l-1;
     n1=1+(rand()%num+1)-1;
     n2=1+(rand()%num+1)-1;
     // Intercambio de posiciones
     c = cad2[n1];
     cad2[n1]=cad2[n2];
     cad2[n2] = c;
   }
   else
   {  rrr++;
   }
   //guarda en la lista de soluciones generadas aleatoreamente para la poblacion inicial
   for(i = 0; i < ln-1; i++)
   {     if(cad2[i]!=0)
```

```
        {
          Resultado[i].DemNodo=instancia.DEMAND[cad2[i]];
           Resultado[i].Norigen=cad2[i];
           Resultado[i].vh=0;
           Resultado[i].tiempo=0;
         }
} //for
for(i=0;i<tamanonodo-1;i++)
        {
                CNN[i]=cad2[i];
        }
for(i=0;i<tamanonodo-1;i++)
                {
                j=instancia.SERVICETIME[i+1];
                TablaVENTANA[i].E=instancia.READYTIME[i+1];
                LTS=TablaVENTANA[i].E+instancia.SERVICETIME[i+1];
                TablaVENTANA[i].LTServicio=LTS;
                DELTA=rand()%j;
                TablaVENTANA[i].DELTAR=DELTA;
                ELE=TablaVENTANA[i].E+DELTA;
                TablaVENTANA[i].ele=ELE;
                Resultado[i].tiempo=TablaVENTANA[i].ele;
                }
        int V=0,nod=0,D=0,J,x,cuenta=0,demanda;
        demanda=instancia.C; V=1; J=0;
        for(i=0;i<tamanonodo-1;i++)
        {
            if(((D+Resultado[i].DemNodo)<demanda))
           {
                Resultado[i].vh=V;
                D=D+Resultado[i].DemNodo;
               J=0;
           }
           else
           {
                 if((D+Resultado[i].DemNodo)>demanda)
                 {
                 V++;
                 D=0;
                 Resultado[i].vh=V;
                 D=D+Resultado[i].DemNodo;
                 }
                 else
                 {
                  D=D+Resultado[i].DemNodo;
                  Resultado[i].vh=V;
                 J=0;
                 }
            }
        }
        //ordena en orden ascendente de vehiculo 1,2,3...10
       for(x=0;x<tamanonodo-2;x++)
        {
```

```
                    for(j=0;j<tamanonodo-2;j++)
                    {
                    if(Resultado[j].vh>Resultado[j+1].vh)
                    {
                    MTAUX[j].dem=Resultado[j].DemNodo;
                    MTAUX[j].nodo=Resultado[j].Norigen;
                    MTAUX[j].tiempo=Resultado[j].tiempo;
                    MTAUX[j].vh=Resultado[j].vh;
                    Resultado[j].DemNodo=Resultado[j+1].DemNodo;
                    Resultado[j].Norigen=Resultado[j+1].Norigen;
                    Resultado[j].tiempo=Resultado[j+1].tiempo;
                    Resultado[j].vh=Resultado[j+1].vh;
                    Resultado[j+1].DemNodo=MTAUX[j].dem;
                    Resultado[j+1].Norigen=MTAUX[j].nodo;
                    Resultado[j+1].tiempo=MTAUX[j].tiempo;
                    Resultado[j+1].vh=MTAUX[j].vh;
                    }
                    }
   }
   //ORDENA EN BASE AL TIEMPO
   for(x=0; x<tamanonodo-1;x++)
          {
          for(j=0;j<tamanonodo-1;j++)
          {

          if((Resultado[j].tiempo>Resultado[j+1].tiempo)&&(Resultado[j].vh==Resultado[j+1].v
          h))
          {
          MTAUX[j].dem=Resultado[j].DemNodo;
          MTAUX[j].nodo=Resultado[j].Norigen;
          MTAUX[j].tiempo=Resultado[j].tiempo;
          MTAUX[j].vh=Resultado[j].vh;
          Resultado[j].DemNodo=Resultado[j+1].DemNodo;
          Resultado[j].Norigen=Resultado[j+1].Norigen;
          Resultado[j].tiempo=Resultado[j+1].tiempo;
          Resultado[j].vh=Resultado[j+1].vh;
          Resultado[j+1].DemNodo=MTAUX[j].dem;
          Resultado[j+1].Norigen=MTAUX[j].nodo;
          Resultado[j+1].tiempo=MTAUX[j].tiempo;
          Resultado[j+1].vh=MTAUX[j].vh;
          }
          }
   }

}


void rutakmeans(void)
{
          int i,j,LTS,DELTA,ELE;
          char cadena[300];
          FILE  *archDatos;
          int repetir;
```

```
        float DemandaT=0.0;
        double RecorridoT=0.0,menor=90000,RECORRTOTAL=0;
        srand((unsigned)time(NULL));
        printf("Proporcione el Archivo que contiene la poblacion inicial: ");
        scanf("%s", archivo); rrr++;
        int cuentador=0; char c;

        if((archDatos=fopen(archivo,"rt"))==NULL)
        { printf(" No existe el archivo de datos ==> [%s]%c",archivo,7);
        }
        printf("\n            El archivo es %s", archivo);
        fgets(cadena,300,archDatos);
        for(repetir=0;repetir<tamanonodo-1;repetir++)
        { //fgets(cadena,300,archDatos);
       fscanf(archDatos,"%d\t",&Resultado[repetir].Norigen);
       fscanf(archDatos,"%d\t",&Resultado[repetir].vh);
         Resultado[repetir].DemNodo=instancia.DEMAND[Resultado[repetir].Norigen];
        }

for(i=0;i<tamanonodo-1;i++)
       {
       j=instancia.SERVICETIME[i+1]; //Bench[1].Service;
       TablaVENTANA[i].E=instancia.READYTIME[i+1]; //Bench[i].ReadyT;
       LTS=TablaVENTANA[i].E+instancia.SERVICETIME[i+1]; // Bench[i].Service;
       TablaVENTANA[i].LTServicio=LTS;
       DELTA=rand()%j;
       TablaVENTANA[i].DELTAR=DELTA;
       ELE=TablaVENTANA[i].E+DELTA;
       TablaVENTANA[i].ele=ELE;
       Resultado[i].tiempo=TablaVENTANA[i].ele;
       }
       int V=0,nod=0,D=0,J,x,cuenta=0,demanda;
       demanda=instancia.C; V=1; J=0;
      for(i=0;i<tamanonodo-1;i++)
      {
          if(((D+Resultado[i].DemNodo)<demanda))
                 {
                 D=D+Resultado[i].DemNodo;
                J=0;
                 }
                 else
                 {
                 if((D+Resultado[i].DemNodo)>demanda)
                 {
                 V++;
                 D=0;
                 Resultado[i].vh=V;
                 D=D+Resultado[i].DemNodo;
                 }
                 }
      }
   //ordena en orden ascendente de vehiculo 1,2,3...10
   for(x=0;x<tamanonodo-2;x++)
```

```
        {
                for(j=0;j<tamanonodo-2;j++)
                {
                if(Resultado[j].vh>Resultado[j+1].vh)
                {
                MTAUX[j].dem=Resultado[j].DemNodo;
                MTAUX[j].nodo=Resultado[j].Norigen;
                MTAUX[j].tiempo=Resultado[j].tiempo;
                MTAUX[j].vh=Resultado[j].vh;
                Resultado[j].DemNodo=Resultado[j+1].DemNodo;
                Resultado[j].Norigen=Resultado[j+1].Norigen;
                Resultado[j].tiempo=Resultado[j+1].tiempo;
                Resultado[j].vh=Resultado[j+1].vh;
                Resultado[j+1].DemNodo=MTAUX[j].dem;
                Resultado[j+1].Norigen=MTAUX[j].nodo;
                Resultado[j+1].tiempo=MTAUX[j].tiempo;
                Resultado[j+1].vh=MTAUX[j].vh;
                }
                }
   }
   //ORDENA EN BASE AL TIEMPO
   for(x=0; x<tamanonodo-1;x++)
        {
        for(j=0;j<tamanonodo-1;j++)
        {

        if((Resultado[j].tiempo>Resultado[j+1].tiempo)&&(Resultado[j].vh==Resultado[j+1].v
        h))
         {
        MTAUX[j].dem=Resultado[j].DemNodo;
        MTAUX[j].nodo=Resultado[j].Norigen;
        MTAUX[j].tiempo=Resultado[j].tiempo;
        MTAUX[j].vh=Resultado[j].vh;
        Resultado[j].DemNodo=Resultado[j+1].DemNodo;
        Resultado[j].Norigen=Resultado[j+1].Norigen;
        Resultado[j].tiempo=Resultado[j+1].tiempo;
        Resultado[j].vh=Resultado[j+1].vh;
        Resultado[j+1].DemNodo=MTAUX[j].dem;
        Resultado[j+1].Norigen=MTAUX[j].nodo;
        Resultado[j+1].tiempo=MTAUX[j].tiempo;
        Resultado[j+1].vh=MTAUX[j].vh;
        }
        }
    }

}


void ordenaVEC()
{

 int V=0,i,x,j,repetir;
  for(i=0;i<vecinos+1;i++)
```

```
{
 //asigna temporalmente el ready time como tiempo
  for(repetir=0;repetir<tamanonodo-1;repetir++)
  {  //asigna temporalmente el ready time como tiempo
        MTVEC[i][repetir].tiempo=instancia.READYTIME[Resultado[repetir].Norigen];
  }
 //calcula demandas
 int nod=0,D=0,J,cuenta=0,demanda;
 demanda=instancia.C; V=1; J=0;
 for(x=0;x<tamanonodo-1;x++)
          {
            if(((D+MTVEC[i][x].dem)<demanda))
                    {
                   MTVEC[i][x].vh=V;
                   D=D+MTVEC[i][x].dem;
                   J=0;
                    }
                    else
                    {
                              if((D+MTVEC[i][x].dem)>demanda)
                              {
                                V++;
                                D=0;
                                MTVEC[i][x].vh=V;
                                D=D+MTVEC[i][x].dem;
                              }
                              else
                              {
                               D=D+MTVEC[i][x].dem;
                               MTVEC[i][x].vh=V;
                              J=0;
                              }
                    }
          } //fin for x


  //ordena en orden ascendente de vehiculo 1,2,3...10
  for(x=0;x<tamanonodo-2;x++)
         {
                   for(j=0;j<tamanonodo-2;j++)
                   {
                   if(MTVEC[i][j].vh>MTVEC[i][j+1].vh)
                   {
                   MTAUX[j].dem=MTVEC[i][j].dem;
                  MTAUX[j].nodo=MTVEC[i][j].nodo;
                   MTAUX[j].tiempo=MTVEC[i][j].tiempo;
                   MTAUX[j].vh=MTVEC[i][j].vh;
                   MTVEC[i][j].dem=MTVEC[i][j+1].dem;
                   MTVEC[i][j].nodo=MTVEC[i][j+1].nodo;
                   MTVEC[i][j].tiempo=MTVEC[i][j+1].tiempo;
                  MTVEC[i][j].vh=MTVEC[i][j+1].vh;
                  MTVEC[i][j+1].dem=MTAUX[j].dem;
                   MTVEC[i][j+1].nodo=MTAUX[j].nodo;
```

```
                MTVEC[i][j+1].tiempo=MTAUX[j].tiempo;
                MTVEC[i][j+1].vh=MTAUX[j].vh;
                }
                }
}

////////////////////////////////////////////////////////////////////////
//para asignar el tiempo que le corresponde
////////////////////////////////////////////////////////////////////////
 //TIMEWINDOWS();
 int R=0,R2=0,RFIN=9999,ok=0;
 for(j=0;j<tamanonodo-1;j++)
        {
         if(MTVEC[i][j].vh==MTVEC[i][j+1].vh)
        {
         R=R+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
        for(x=j;x<tamanonodo-1;x++)
        {
          if(MTVEC[i][x].tiempo!=0)
          {   if((R<MTVEC[i][x].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][x].vh))//&&(R<RFIN))
                    {
                     MTVEC[i][j].tiempo=R;
                      RFIN=MTVEC[i][x].tiempo;
                      ok=x;
                      break;
                    }
                   else
                    {

                  if((R>MTVEC[i][x].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][x].vh))//&&(R>RFIN
                  ))
                            {
                       R=instancia.DUEDATE[MT[i][x].nodo];
                      RFIN=R;
                      //mueve el mayor tiempo a donde le corresponde
                      MTAUX[x].dem=MTVEC[i][x].dem;
                      MTAUX[x].nodo=MTVEC[i][x].nodo;
                       MTAUX[x].tiempo=MTVEC[i][x].tiempo;
                       MTAUX[x].vh=MTVEC[i][x].vh;
                      MTVEC[i][x].dem=MTVEC[i][j].dem;
                      MTVEC[i][x].nodo=MTVEC[i][j].nodo;
                      MTVEC[i][x].tiempo=MTVEC[i][j].tiempo;
                      MTVEC[i][x].vh=MTVEC[i][j].vh;
                      MTVEC[i][j].dem=MTAUX[x].dem;
                       MTVEC[i][j].nodo=MTAUX[x].nodo;
                       MTVEC[i][j].tiempo=MTAUX[x].tiempo;
                       MTVEC[i][j].vh=MTAUX[x].vh;
                       ok=999;
                       break;
                            }
                   }//fin else
                   } //fin si !=0
                  }//fin for
```

```
                }
          else if(MTVEC[i][j].vh!=MTVEC[i][j+1].vh)
          {
        R=R+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
        //Resultado[i].tiempo=R;
        R=0; RFIN=9999;
          }
         }//fin for

    for(j=0;j<tamanonodo-1;j++)
          {
         if((MTVEC[i][j].tiempo==0)&&(MTVEC[i][j].vh==MTVEC[i][j+1].vh))
          {
           R=MTVEC[i][j-1].tiempo+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
           MTVEC[i][j].tiempo=R;
          }

          if((MTVEC[i][j].vh!=MTVEC[i][j+1].vh)&&(MTVEC[i][j].tiempo==0))
          {
           R=MTVEC[i][j-1].tiempo+instancia.SERVICETIME[MTVEC[i][j+1].nodo];
        MTVEC[i][j].tiempo=R;
         }
  }//fin for

  for(x=0; x<tamanonodo-1;x++)
  {
         for(j=0;j<tamanonodo-1;j++)
         {
         if((MT[i][j].tiempo>MTVEC[i][j+1].tiempo)&&(MTVEC[i][j].vh==MTVEC[i][j+1].vh))
           {
                  MTAUX[j].dem=MTVEC[i][j].dem;
                 MTAUX[j].nodo=MTVEC[i][j].nodo;
                  MTAUX[j].tiempo=MTVEC[i][j].tiempo;
                  MTAUX[j].vh=MTVEC[i][j].vh;
                  MTVEC[i][j].dem=MTVEC[i][j+1].dem;
                  MTVEC[i][j].nodo=MTVEC[i][j+1].nodo;
                  MTVEC[i][j].tiempo=MTVEC[i][j+1].tiempo;
                  MTVEC[i][j].vh=MTVEC[i][j+1].vh;
                  MTVEC[i][j+1].dem=MTAUX[j].dem;
                  MTVEC[i][j+1].nodo=MTAUX[j].nodo;
                  MTVEC[i][j+1].tiempo=MTAUX[j].tiempo;
                  MTVEC[i][j+1].vh=MTAUX[j].vh;
           }
         }
  }
  }//fin del i
}
```