PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# DESARROLLO DE UN SISTEMA MULTIAGENTE PARA PARALELIZAR EL PROBLEMA DEL TRASPORTE DE PASAJEROS CON VENTANAS DE TIEMPO (DARPTW) RESUELTO CON ALGORITMOS GENETICOS

**FERNANDA VALERIA NÚÑEZ GONZÁLEZ**

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

Octubre 2011

Pontificia Universidad Católica de Valparaíso – Chile
Facultad de Ingeniería
Escuela de Ingeniería Informática

# DESARROLLO DE UN SISTEMA MULTIAGENTE PARA PARALELIZAR EL PROBLEMA DEL TRASPORTE DE PASAJEROS CON VENTANAS DE TIEMPO (DARPTW) RESUELTO CON ALGORITMOS GENETICOS

**FERNANDA VALERIA NÚÑEZ GONZÁLEZ**

Profesor Guía: **Claudio Cubillos Figueroa**

Carrera: **Ingeniería Civil Informática**

Octubre 2011

*Dedico este trabajo a mi familia y amigos,*
*principalmente a mis padres, por su gran apoyo*
*durante todas las etapas de mi desarrollo profesional.*

# Índice

# Glosario de Términos

Allele: Forma en que puede presentarse un gen en un determinado locus de cromosomas homólogos.

Fitness: Valor de calidad que se da a un genotipo en relación a su habilidad para sobrevivir y reproducirse.

Locus: Punto de un cromosoma ocupado por un gen.

Clustering: Procedimiento de agrupación de un conjunto de datos o casos de acuerdo a algún criterio.

# Lista de Abreviaturas o Siglas

| | |
|---|---|
| ACL | : Agent Communication Language. |
| AIP | : Agent Interaction Protocol. |
| DARPTW | : Dial-a-Ride Problem with Time Windows. |
| DARP | : Dial-a-Ride Problem. |
| DRT | : Direct Ride Time. |
| DRTS | : Demand Responsive Transport System. |
| GA | : Genetic Algorithm. |
| JADE | : Java Agents Development Environment. |
| LLGA | : Linkage Learning Genetic Algorithm. |
| MA | : Multi-agent Architecture. |
| MAS | : Multi-agent System. |
| MRT | : Maximum Ride Time. |
| PASSI | : Process for Agent Societies Specification and Implementation. |
| PGA | : Parallel Genetic Algorithm. |
| PDP | : Pickup and Delivery Problem. |
| TSP | : Travelling Salesman Problem. |
| TW | : Time Window. |
| VRP | : Vehicle Routing Problem. |

# Lista de Figuras

# Lista de Tablas

# Resumen

El Dial-a-Ride Problem with Time Windows, es un problema de múltiples objetivos donde se deben satisfacer un conjunto de peticiones de transporte de personas desde un lugar de origen a uno de destino mediante una red de locaciones que constituyen una ruta. El problema posee distintos tipos de restricciones, siendo las más relevantes las ventanas de tiempo y la capacidad de los vehículos. Esta dificultad del problema ha motivado la utilización de algoritmos evolutivos para su resolución y otro tipo de heurísticas.

El presente trabajo pretende resolver el DARPTW mediante la utilización de algoritmos genéticos paralelos, implementando un sistema multiagente para permitir su resolución en un ambiente distribuido.

**PALABRAS CLAVE:** Algoritmo Genético, Sistema Multiagente, DARPTW, paralelización.

# Abstract

The Dial-a-Ride Problem with Time Windows, is a multi-objective problem which must satisfy a set of requests for transportation of persons from a place of origin to a destination through a network of locations that constitute a path. Furthermore, the problem has various types of restrictions being the most relevant the time windows and vehicle capacity. This difficulty has motivated the use of evolutionary algorithms for resolution and other heuristics.

This work addresses the DARPTW using parallel genetic algorithms, implementing a multi-agent system to allow its resolution in a distributed environment.



**KEY WORDS:** Gentic Algorithm, Multi-Agent System, DARPTW, parallelization.

# 1 Presentación del tema

## 1.1 Introducción

En proyectos anteriores se plantea el diseño de un Sistema Multiagente (MAS) [14] para resolver el Dynamic Dial-a-Ride Problem (D-DARP) y por otra parte se proponen algoritmos genéticos para resolver el problema de transporte de pasajeros con ventanas de tiempo (DARPTW) [18]. Sin embargo, no existe integración entre ambas implementaciones y las heurísticas utilizadas para la resolución del problema no han sido paralelizadas.

Considerando como oportunidad la integración de ambos proyectos y la ventaja que supone paralelizar las heurísticas de resolución en problemas de tiempo real, se propone el estudio y revisión de estos, y una implementación completa de un MAS que permita resolver el DARPTW de forma paralela utilizando como heurística de resolución los algoritmos genéticos. Para el desarrollo de esta propuesta se utilizará la metodología PASSI [15], que es una metodología de desarrollo de MAS que pretende participar en cada una de las etapas de desarrollo del sistema.

Dentro de la comunidad de Investigación de Operaciones, el estudio de sistemas de transporte, tanto en sus características como en su planificación, ha sido de gran relevancia desde hace muchos años. Algunos de los que han ofrecido mayores desafíos en el contexto de la investigación son los que se pueden clasificar como “en respuesta a la demanda” (Demand Responsive Transport Systems-DRTS). Estos se caracterizan por el hecho que se realiza una solicitud previa a la ejecución del servicio, lo que se conoce como una “solicitud avanzada”, por ejemplo cuando un cliente solicita el servicio por medio de una llamada telefónica. Se trata de servicios cuyas rutas y horarios son variables, además que funcionan principalmente de puerta a puerta a diferencia de los sistemas de transporte más tradicionales que lo hacen de parada a parada. Desde los años 70 que son usados, en particular para el transporte de adultos mayores y personas con discapacidad, sin embargo, gracias a nuevas tecnologías que han ido apareciendo en los últimos años, su consideración y alcances han aumentado de manera importante.

Las principales dificultades al abordar este tipo de sistemas se generan en virtud de la naturaleza de las restricciones y parámetros críticos que ellos presentan. Por ejemplo, se han desarrollado estudios enfocados a comprender y predecir las consecuencias que generan distintas políticas de administración en estos sistemas de transporte, en base al análisis de los requerimientos de los pasajeros y el costo de los operadores asociados al problema. También vía simulación se ha demostrado de forma cuantitativa la dependencia entre la productividad y costos de estos sistemas con dos factores importantes como son el tamaño de las ventanas de tiempo (mencionadas en lo que sigue) y el tipo de estrategia (centralizada o descentralizada) que se utilice.

En términos de problema de optimización, un DRTS puede representarse en el Dial-a-Ride Problem (DARP) [17] o el problema de transporte de pasajeros. Consiste en buscar la mejor forma de transportar un conjunto de pasajeros, distribuidos espacialmente en una red de

locaciones, en base a una serie de restricciones de distinto tipo. Entre estas destacan la capacidad limitada de cada vehículo y la consideración de ventanas de tiempo o time Windows (TW), que corresponden a un intervalo válido de tiempo en el cual un usuario puede ser recogido o entregado en la locación origen o destino respectivamente que especifique. El objetivo es optimizar los factores vinculados al sistema de transporte (número de vehículos requeridos, costos de viaje) y al servicio entregado a los usuarios (tiempo de espera, tiempo de transporte). DARP es una versión particular de una familia de problemas de transporte, en los que se consideran (en orden de generalidad respectivo) el travelling salesman problem, el vehicle routing problem y el pickup and delivery problem. DARPTW es considerado un problema de tipo NP-Hard, principalmente por la restricción que imponen las TW. Por ello, en términos prácticos, este problema suele ser abordado con la utilización de heurísticas para encontrar soluciones aproximadas, bajo las distintas variantes que pueda presentar.

Una de las herramientas usadas ampliamente en problemas de transporte son los Algoritmos Genéticos (GA) [2]. Tras su aparición en el escenario científico, este tipo de algoritmos ha recibido una gran aceptación por su eficiencia para resolver problemas de distinta complejidad. Sin embargo, esta eficiencia depende mucho del tipo de problema planteado, junto con la estrategia que adopte el algoritmo para abordarlo. Hasta nuestros días, son numerosos los estudios que se han realizado en el área respecto a las dificultades que el GA básico puede tener al abordar problemas de optimización de alta complejidad.

## 1.2 Objetivos del proyecto

### 1.2.1 Objetivo General

Desarrollar un sistema multiagente para resolver el problema de transporte de pasajeros con ventanas de tiempo (DARPTW) de forma paralela, utilizando Algoritmos Genéticos.

### 1.2.2 Objetivos Específicos

- Comprender el problema de transporte de pasajeros con ventanas de tiempo y definir los principales conceptos asociados.
- Comprender las variantes de Algoritmos Genéticos utilizadas para resolver el problema de transporte de pasajeros con ventanas de tiempo.
- Realizar la migración del código estudiado en la investigación anterior, de lenguaje de programación C# a Java.
- Diseñar un sistema multiagente para paralelizar el problema, utilizando metodología PASSI, tomando en cuenta las interacciones y comportamientos entre los agentes.
- Desarrollar un sistema multiagente de acuerdo al diseño realizado.
- Diseñar escenarios de prueba para validar el sistema propuesto utilizando benchmarks disponibles en la literatura.

## 1.3 Metodología de trabajo

Para el desarrollo del trabajo, se utilizará una metodología de investigación científica clásica. Esta se puede describir en base a las siguientes etapas (ver Figura 1.1):

- El desarrollo del proyecto comienza con la investigación y elección del tema central. En base a este se debe realizar el planteamiento del problema, luego se definen los objetivos del trabajo (general y específicos). Además, es importante delimitar el tema definiendo el alcance del proyecto (1, 2 y 3).
- A partir del planteamiento del problema, se desarrolla el marco teórico, donde se identifican las variables involucradas en el problema y se definen los principales términos asociados a este. Aquí se contempla la revisión de la literatura para la obtención, selección y la extracción de antecedentes relevantes (4 y 5).
- Finalmente se propone una metodología para desarrollar la solución al problema planteado, donde se deben considerar el procesamiento de datos, la definición y recolección de datos de prueba para validar la solución propuesta. La metodología debe considerar, para todo el proceso, la elaboración y presentación de informes de investigación adecuados (6 y 7).

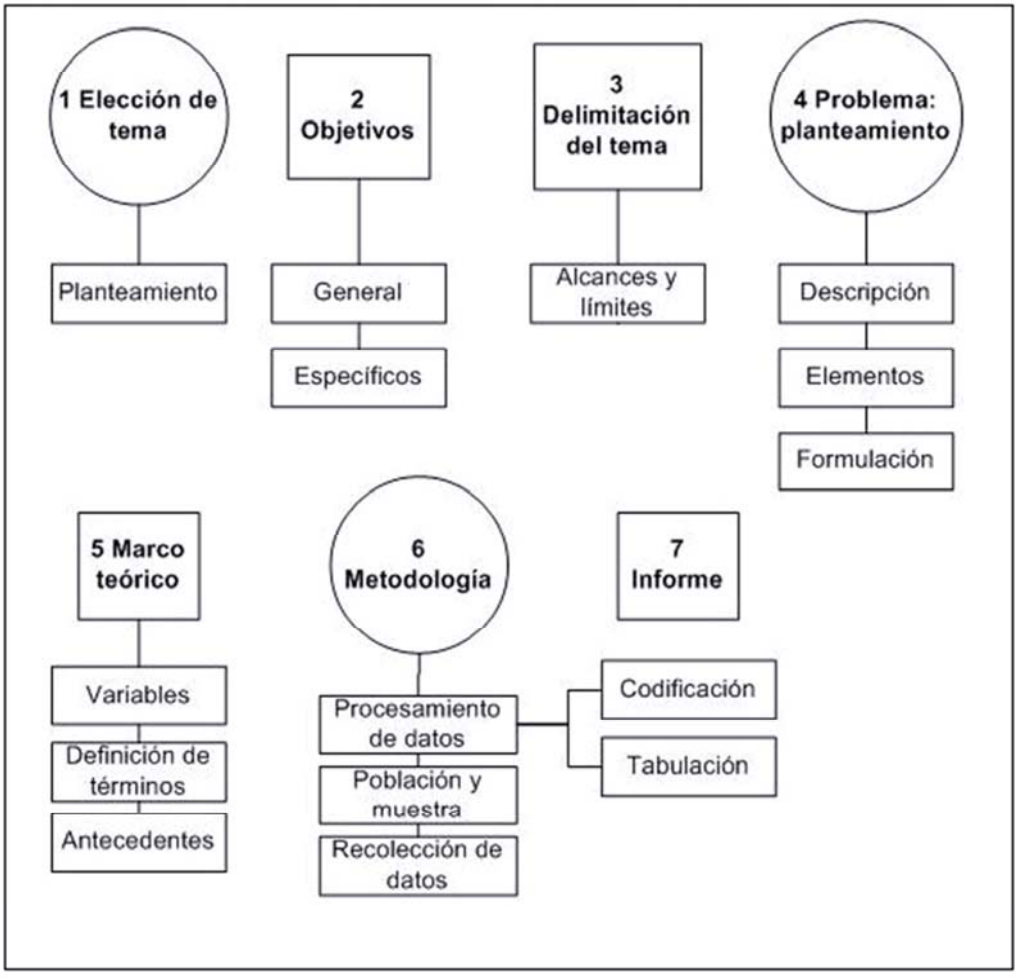


Figura 1.1 Metodología de trabajo.

## 1.4 Plan de trabajo

A continuación se presenta un plan de trabajo tentativo para el desarrollo del presente proyecto.

| ETAPA | ACTIVIDADES | FECHA INICIO | FECHA TERMINO |
|---|---|---|---|
| Proyecto 1 | Obtener estado del arte sobre sistemas multiagentes. | 22 de Marzo | 26 de Marzo |
| | Obtener estado del arte sobre los problemas de transporte y el problema de transporte de pasajeros (DARP). | 27 de Marzo | 29 de Marzo |
| | Obtener estado del arte sobre el problema de transporte de pasajeros con ventanas de tiempo (DARPTW) y sus variantes. | 30 de Marzo | 2 de Abril |
| | Obtener estado del arte sobre Algoritmos Genéticos. | 3 de Abril | 6 de Abril |
| | Estudiar la heurística de resolución a este problema utilizando Algoritmos Genéticos, ya propuesta en el proyecto anterior mencionado en la descripción del problema. | 7 de Abril | 11 de Abril |
| | Definir decisiones con respecto a la resolución del problema, además decidir sobre el protocolo, la plataforma y el lenguaje de programación. | 12 de Abril | 16 de Abril |
| | Entrega de avance de Proyecto 1. | | 23 de Abril |
| | Obtener el Modelo de Requerimientos del Sistema. | 26 de Abril | 30 de Abril |
| | Obtener Modelo de Sociedad de Agentes. | 1 de Mayo | 5 de Mayo |
| | Obtener Modelo de Implementación de Agentes. | 6 de Mayo | 13 de Mayo |
| | Revisar el modelado final propuesto para el sistema, agregando diagramas o algún otro tipo de especificación. | 14 de Mayo | 15 de Mayo |

| | | | |
|---|---|---|---|
| | Obtener el diseño final del sistema multiagente para su futura implementación. | 16 de Mayo | 18 de Mayo |
| | Migración del código del algoritmo genético de C# a Java. | 19 de Mayo | 26 de Mayo |
| | Implementación de un primer prototipo. | 27 de Mayo | 1 de Junio |
| | Entrega de informe final de Proyecto 1. | | 2 de Junio |
| Proyecto 2 | Migración del código de C# a Java. | 30 de Agosto | 20 de Septiembre |
| | Entrega avance Proyecto 2. | | 24 de Septiembre |
| | Desarrollo del sistema multiagente. | 27 de Septiembre | 15 de Octubre |
| | Integración del código migrado y el sistema multiagente y pruebas. | 17 de Octubre | 30 de Octubre |
| | Revisión de funcionamiento esperado y correcciones. | 1 de Noviembre | 6 de Noviembre |
| | Conclusión de documentos finales de Proyecto 2. | 7 de Noviembre | 11 de Noviembre |
| | Entrega final Proyecto 2. | | 12 de Noviembre |

Tabla 1.1 Planificación tentativa.

# 2 Algoritmos Genéticos

## 2.1 Reseña histórica

En la década de 1950 y 1960 varios científicos estudiaron de manera independiente los sistemas evolutivos, con la idea de que la evolución podría ser utilizada como una herramienta de optimización para problemas de ingeniería. La idea era evolucionar una población de soluciones candidatas a un problema determinado, utilizando operadores inspirados en la variación genética natural y la selección natural.

Durante estas décadas, varios otros más desarrollaron algoritmos inspirados en la evolución para optimización y aprendizaje automático. Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962), y Reed, Toombs, y Baricelli (1967), trabajaron en esta área, aunque se le ha dado una pequeña o casi nula atención comparada con la que se ha dado a las estrategias de evolución, programación evolutiva y algoritmos genéticos [40].

Los Algoritmos Genéticos (GAs) fueron propuestos por John Holland en 1960 y fueron desarrollados por Holland, sus alumnos y colegas de la Universidad de Michigan en las décadas de 1960 y 1970.

En contraste con las estrategias evolutivas y la programación evolutiva, el objetivo original de Holland no era diseñar algoritmos para resolver problemas específicos, sino estudiar formalmente el fenómeno de la adaptación como ocurre en la naturaleza y desarrollar formas en que los mecanismos de adaptación natural pudiesen ser importados en los sistemas informáticos.

El libro “Adaptation in Natural and Artificial Systems” escrito por Holland en 1975, presentó el algoritmo genético como una abstracción de la evolución biológica y proporcionó un marco teórico para la adaptación en el GA. El algoritmo genético de Holland es un método para mover desde una población de “cromosomas” (por ejemplo, cadenas de unos y ceros, o “bits”) a una nueva población a través de un tipo de “selección natural”, junto con operadores inspirados en la genética tales como cruzamiento, mutación e inversión. Cada cromosoma se compone de “genes” (por ejemplo bits), y cada uno de ellos puede ser una instancia de un “allele” [1]. El operador de selección se encarga de escoger los individuos que participaran en la formación de la nueva población, el de cruce intercambia las propiedades de los individuos y el de mutación produce cambios aleatorios en la población de tal forma que se amplíe el espacio de búsqueda. Holland también introdujo el operador inversión que invierte el orden de una parte del cromosoma.

## 2.2 Conceptos básicos

### 2.2.1 Algoritmo Genético

Los algoritmos genéticos (GA) son algoritmos de búsqueda basadas en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya randomizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana [2].

Una definición más completa está dada por John Koza: El algoritmo genético es un algoritmo matemático altamente paralelo que transforma un conjunto (población) de objetos matemáticos individuales, cada uno de los cuales se asocia con una aptitud, en una población nueva, es decir en la siguiente generación, usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas en las que destaca la recombinación sexual. Los objetos matemáticos corresponden a individuos de la población, los cuales típicamente son cadenas de longitud fija que se ajustan al modelo de las cadenas de cromosomas. [3]

Los algoritmos genéticos difieren de otros métodos de búsqueda, ya que mantienen siempre un conjunto de soluciones llamadas población. Los miembros de la población son los cromosomas que están representados originalmente por cadenas de bits. La primera población se inicializa al azar y evoluciona en generaciones. En cada generación la población es afectada por los operadores genéticos y mecanismos de selección. Los operadores genéticos, tales como los operadores de cruzamiento y mutación, proporcionan el flujo de información entre los cromosomas, mientras que la selección promueve la supervivencia de los cromosomas más aptos. Una función de calidad evalúa los cromosomas individuales. El mecanismo de selección suele ser una combinación de la función de calidad con cierta probabilidad.

### 2.2.2 Algoritmo Genético Simple o Canónico

En un algoritmo genético canónico, cada miembro de una población corresponde a una cadena binaria, que se denomina “genotipo” o “cromosoma”. Cada uno de ellos, luego de generar la población inicial que en la mayoría de los casos se hace de manera aleatoria, será evaluado y se le asignará un valor de “calidad” (fitness).

La función de evaluación proporciona una medida de rendimiento con respecto a un conjunto de parámetros determinados. Por otro lado, la función de calidad transforma esta medida de rendimiento en una asignación de posibilidades de reproducción. La evaluación de un individuo es independiente de la evaluación de cualquier otro, mientras que la calidad de un individuo siempre es definida con respecto a otros miembros de la población.

Se define la calidad por la expresión $f_i/f$, donde $f_i$ corresponde a la evaluación asociada a la cadena i y $f$ el promedio de los resultados de evaluación en la población. La probabilidad de que un individuo de la población actual se copie y pase a la generación intermedia es proporcional a su calidad, definida anteriormente.

La ejecución de un algoritmo genético se puede considerar como un proceso de dos etapas. En la primera se le aplica un proceso de selección a la población actual, con lo cual se genera una población intermedia. Luego, a esta última se le realizan operaciones de recombinación (o cruzamiento) y mutación para generar la próxima población. El paso de una población a otra mediante el proceso anterior corresponde a una generación.

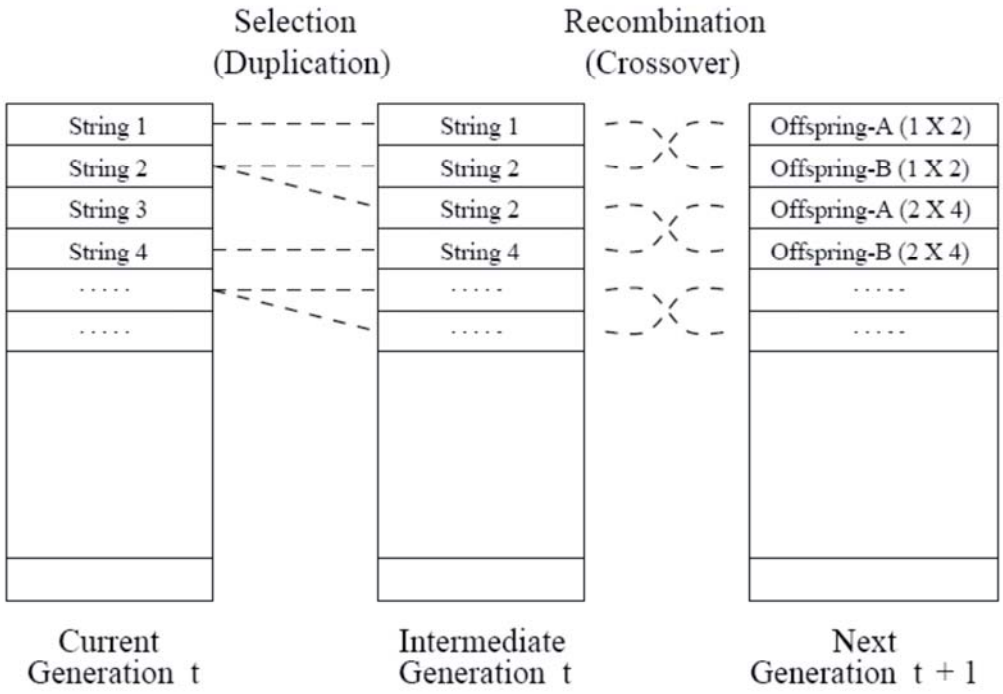


Figura 2.1 Esquema de Selección y Cruzamiento [36].

La Figura 2.1 muestra la asignación de los individuos a los slots adyacentes durante el proceso de selección, los que pueden ser asignados de manera aleatoria con el fin de mezclar la población intermedia. El proceso de mutación puede realizarse luego del cruzamiento.

El pseudocódigo presentado en la Figura 2.2 representa el algoritmo genético simple o canónico. Se necesita una codificación o representación del problema, que se adecue a éste.

Además, una función de ajuste o adaptación al problema, que asignará un número real a cada solución codificada posible.

En la ejecución del algoritmo son seleccionados los padres para la reproducción. Luego, éstos se cruzarán generando hijos, sobre los cuales actuará un operador de mutación. El resultado de lo anterior será un conjunto de individuos que representarán las posibles soluciones al problema, y que formarán parte de la siguiente población en la evolución del algoritmo genético.

```
BEGIN /* Algoritmo Genético Simple */
 Generar una población inicial.
 Computar la función de evaluación de cada individuo.
 WHILE NOT Terminado DO
       BEGIN /* Producir nueva generación */
             FOR Tamaño población/2 DO
             BEGIN /* Ciclo Reproductivo */
                   Seleccionar dos individuos de la anterior
                   generación, para el cruce (probabilidad de selección
                   proporcional a la función de evaluación del
                   individuo).
                   Cruzar con cierta probabilidad los dos individuos
                   obteniendo dos descendientes.
                   Mutar los dos descendientes con cierta probabilidad.
                   Computar la función de evaluación de los dos
                   descendientes mutados.
                   Insertar los dos descendientes mutados en la nueva
                   generación.
             END
             IF    la población ha convergido THEN
                   Terminado:= TRUE
       END
END
```

Figura 2.2 Pseudocódigo de un Algoritmo Genético Simple.

### 2.2.3 Codificación

Los cromosomas deben contener la información respecto a la solución que representan, para esto se utiliza la codificación. Esta se puede realizar de diversas formas, la más utilizada es a través de una cadena de números binarios (unos y ceros), sin embargo también puede realizarse mediante números enteros o incluso cadenas de caracteres.

El criterio de elección de la codificación dependerá principalmente del problema que se desee resolver, ya que podría suceder que para la resolución de un caso de estudio, el uso de un tipo de codificación sea más óptimo que para otro en que si se utiliza este mismo, pueda perjudicar su solución.

### 2.2.4 Selección

La selección es el proceso que se encarga de escoger a los individuos más capacitados, en relación a su calidad, para formar parte de otra generación y engendrar nuevos individuos.

- **Selección proporcional a la calidad** (*fitness proportionate selection*): también conocida como selección de la ruleta (*ruolette-wheel selection*), se basa en asignar una probabilidad de selección a cada individuo de acuerdo a su calidad. Esto permite que aunque los individuos con alta calidad tengan altas posibilidades, no necesariamente serán seleccionados. Lo mismo en el caso de los individuos con baja calidad, igualmente tienen posibilidades de ser seleccionados. Esto es favorable para la variedad y para evitar la convergencia prematura.
- **Selección por torneo** (*tournament selection*): se trata de seleccionar un conjunto de individuos en la población y escoger los mejores entre ellos para ser parte de futuras generaciones y procesos de recombinación. En este tipo de selección, el tamaño del conjunto seleccionado es un factor relevante. En el torneo, se va evaluando con una probabilidad $p$ cada individuo en orden de calidad, si es seleccionado o no para recombinación. Una selección por torneo se define determinista si $p = 1$.
- **Muestreo estadístico de resto** (*remainder stochastic sampling*): Considerando la expresión de calidad $f_i / F$ de la sección 2.2.2 cuando este valor es mayor que 1, la parte entera de este número indica cuantas copias del cromosoma respectivo son copiadas a la población intermedia. Todos los cromosomas pondrán copias en la población intermedia con probabilidad equivalente a la parte fraccionaria de $f_i / F$. Por ejemplo, si este valor es 1.36, se coloca una copia y hay una probabilidad de 0.36 de que coloque otra.

### 2.2.5 Cruzamiento o Recombinación

El cruzamiento consiste en el intercambio de material genético entre dos individuos para generar un tercer individuo.

El cruce basado en un punto es el más tradicional. En éste los individuos seleccionados como padres se recombinan por medio de una sección de punto de cruce, para intercambiar posteriormente las secciones que se encuentran a la derecha de dicho punto.

Se han investigado otros operadores de cruce, que toman en cuenta más de un punto de cruce. Según la investigación realizada por Jong [4] el cruce basado en dos puntos representaba una mejora, mientras que al añadir más puntos de cruce no se beneficiaba el comportamiento del algoritmo. Si bien, la ventaja de tener más de un punto de cruce radica en que el espacio de búsqueda puede ser explorado de manera más fácil, su desventaja es que aumenta la probabilidad de ruptura de buenos esquemas.

El cruzamiento en dos puntos, como se muestra en la Figura 2.3, consiste en la elección de dos puntos en los cromosomas y los segmentos entre los puntos seleccionados se combinan en los nuevos individuos.

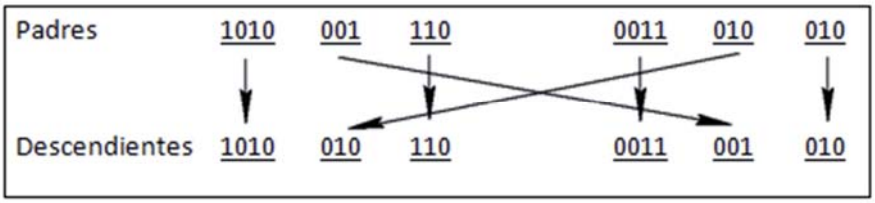


Figura 2.3 Cruzamiento en dos puntos [Elaboración propia].

En el cruzamiento uniforme, mostrado en la Figura 2.4, cada gen en la descendencia se crea copiando el gen correspondiente de uno de los padres, el cual se escoge de acuerdo a una "máscara de cruce" que se genera aleatoriamente. Según el valor de dicha "máscara", el gen es copiado del primer padre, o bien del segundo. Una variante a este método de cruzamiento es el medio cruzamiento uniforme, en el cual se intercambian exactamente la mitad de los genes distintos.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Máscara de cruce | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| Padre 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| | ↓ | | | ↓ | | | ↓ |
| Descendiente | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | ↑ | ↑ | | ↑ | ↑ | |
| Padre 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

Figura 2.4 Cruzamiento Uniforme [Elaboración propia].

### 2.2.6 Mutación

En general, una mutación es un suceso poco común que contribuye a la diversidad genética de la especie. De igual manera, en un algoritmo genético su probabilidad de ocurrencia también es muy baja y su función principal es mantener la diversidad de la población y evitar que exista convergencia prematura en la ejecución del algoritmo, además de permitir explorar otros sectores en el espacio de búsqueda. Las formas de realizarla son variadas, se tiene la más sencilla, en la que cada gen muta aleatoriamente con independencia del resto de genes y otras más complejas en las que se debe tomar en cuenta la estructura del problema y la relación entre los distintos genes.

El objetivo de una mutación es producir nuevas soluciones a partir de la modificación de cierto número de genes presentes en una solución ya existente, y así fomentar la variabilidad dentro de la población.

Cuando ya se establece la frecuencia de mutación, se examinan los bits de cada cadena al momento de crear el nuevo individuo a partir de sus padres, lo que normalmente se realiza de manera simultánea al cruzamiento. Si un número generado aleatoriamente se encuentra por debajo de esa probabilidad, se cambiará el bit (de 0 a 1 o de 1 a 0), de lo contrario se dejará tal como está. Dependiendo del número de individuos y de bits por individuo, podrían resultar mutaciones considerablemente extrañas en una sola generación.

## 2.3 Algoritmos Genéticos Paralelos

Los algoritmos genéticos paralelos (PGAs) no difieren mayormente de los algoritmos genéticos tradicionales, su principal diferencia está en la manera en que tratan los operadores y la población. Con ellos es posible tener una mayor cantidad de individuos en una población y realizar las operaciones de evaluación sobre ellos en un tiempo equivalente al que le tomaría a un algoritmo genético tradicional operar con una población mucho menor. Debido a lo anterior se evitan convergencias prematuras, por lo que los individuos que son seleccionados como soluciones óptimas pueden ser mejores que los obtenidos de un algoritmo genético tradicional [5].

Otra de las características que diferencia este tipo de algoritmo genético con el tradicional, es que se pueden agregar nuevos operadores dependiendo del tipo de algoritmo genético paralelo con que se esté trabajando. Este dependerá del problema que se quiere resolver, la forma de enfrentar el desarrollo del algoritmo, las estrategias de resolución planteadas y la topología en que se ejecutará.

Existen tres características que podrían influir en la eficiencia de un algoritmo genético paralelo:

- La topología que define la comunicación entre subpoblaciones.
- El número de individuos a intercambiar (proporción de intercambio)
- La periodicidad con que se intercambian los individuos (intervalos de migración).

La idea principal consiste en dividir la población total en varias subpoblaciones y en cada una de ellas llevar a cabo un algoritmo genético. Cada cierto número de generaciones, se efectúa una migración que corresponde a un intercambio de información entre las subpoblaciones.

La introducción de la migración hace que los modelos de clasificación sean capaces de explotar las diferencias entre las diversas subpoblaciones, obteniéndose así una fuente de diversidad genética. Es importante determinar la tasa de migración, ya que de esto puede depender la convergencia prematura de la búsqueda.

### 2.3.1 Clasificación

Existen tres tipos principales de algoritmos genéticos paralelos [6]: algoritmos maestro-esclavo, algoritmos de grano fino y algoritmos de grano grueso o de múltiples poblaciones.

En un algoritmo maestro esclavo, al igual que en un algoritmo genético simple, existe una sólo una población panmíctica (población en que todos los individuos tienen la misma probabilidad de aparearse y el apareamiento es al azar), pero la evaluación de la calidad es distribuida entre varios procesos. Dado que en este tipo de algoritmos genéticos paralelos, la selección y el cruzamiento consideran la población en su totalidad, también se conoce como GA paralelo global. Los algoritmos genéticos de grano fino son adecuados para computadores masivamente paralelos y consiste en una población espacialmente estructurada. La selección y el apareamiento está restringido a una pequeña vecindad, pero las vecindades se superponen permitiendo algún tipo de interacción entre todos los individuos. El caso ideal es tener sólo un individuo para cada elemento de procesamiento disponible.

Los algoritmos de grano grueso son más sofisticados, ya que consisten en varias subpoblaciones las cuales intercambian individuos ocasionalmente. Este intercambio de individuos es llamado migración y es controlada por varios parámetros. Este tipo de algoritmo paralelo es muy popular, pero también es la clase de algoritmo genético paralelo más difícil de comprender, ya que los efectos de la migración no son totalmente entendibles. Estos algoritmos introducen cambios fundamentales en la operación de los algoritmos genéticos y tienen un comportamiento diferente a los GAs simples.

A continuación se presentan de manera más detallada los distintos tipos de algoritmos genéticos paralelos.

#### 2.3.1.1 Paralelización Maestro-Esclavo

El algoritmo maestro-esclavo (ver Figura 5) utiliza una única población y la evaluación de los individuos y/o la aplicación de operadores está hecha en paralelo. Tal como en el GA en serie, cada individuo puede competir y parearse con algún otro (su selección y apareamiento son globales).

Por lo general, estos algoritmos son implementados como programas maestro-esclavo, donde el maestro almacenará la totalidad de la población y los esclavos evaluarán la calidad y realizarán la selección. La evaluación de los individuos es paralelizada a través de la asignación de una fracción de la población a cada uno de los procesadores disponibles. La comunicación ocurre sólo cuando cada esclavo recibe su subconjunto de individuos a evaluar y cuando retornan los valores de calidad, y al ocurrir esto último es el maestro el encargado de determinar cuál es la población final del problema a través de una última evaluación sobre la población resultante.

Un algoritmo maestro-esclavo síncrono tiene las mismas propiedades que un GA simple, siendo la velocidad su única diferencia. Este para y espera para recibir los valores de

calidad para toda la población antes de pasar a la próxima generación. Por otro lado, también es posible implementar uno asíncrono el que, al contrario del anterior, no se detiene para esperar a los procesadores lentos.

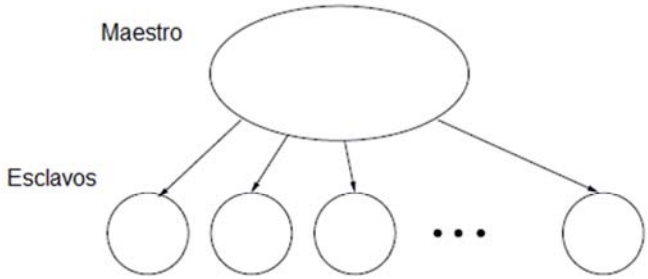


Figura 2.5 Algoritmo Paralelo Maestro-Esclavo [6].

#### 2.3.1.2 Algoritmo Genético Paralelo de Grano Fino

Este algoritmo consta sólo de una población y además de una estructura espacial que limita la interacción entre los individuos, cada uno de los cuales sólo puede competir y aparearse con sus vecinos (ver Figura 2.6). Sin embargo, se permite el solapamiento entre vecindarios para propiciar una interacción leve, entre todos los individuos de la población.

Se han realizado estudios para determinar si el tamaño de los vecindarios podría influir en la presión de selección de los individuos. Sarma y De Jong [7] analizaron los efectos del tamaño y la forma de la vecindad en el mecanismo de selección, y encontraron que la relación entre el radio de los vecindarios y el radio de la totalidad de la población influía en este aspecto. Su análisis cuantificó el tiempo que tardaba en propagarse una buena solución al resto de la población con vecindarios de diferentes tamaños.

Otro estudio, realizado por Schwehm [8] intentó determinar qué tipo de representaciones espaciales podrían dar mejores resultados. Las estructuras probadas fueron un anillo, un toro, un cubo $16 \times 16 \times 16$, un hipercubo $4 \times 4 \times 4 \times 4$ y un hipercubo binario 10-D. Como resultado, el algoritmo que utilizó la estructura de toro convergió de manera más rápida que el resto, pero no se menciona la calidad de las soluciones encontradas.

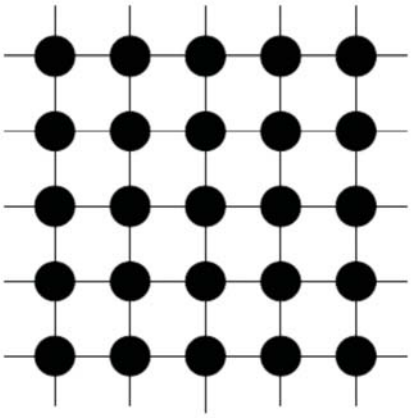

Figura 2.6 Algoritmo Paralelo de Grano Fino [6].

#### 2.3.1.3 Algoritmo Genético Paralelo de Grano Grueso o de Múltiples Poblaciones

Este tipo de algoritmos se caracterizan principalmente por el uso de múltiples poblaciones (ver Figura 2.7) y la migración de individuos entre ellas. En ellos el rango de migración será muy importante para obtener resultados satisfactorios, ya que cada una de las poblaciones evoluciona independientemente.

El buen funcionamiento de estos algoritmos depende de diversos factores, entre los que se encuentran la frecuencia con que se realizan las migraciones y la topología de comunicación que se utilice entre las subpoblaciones.

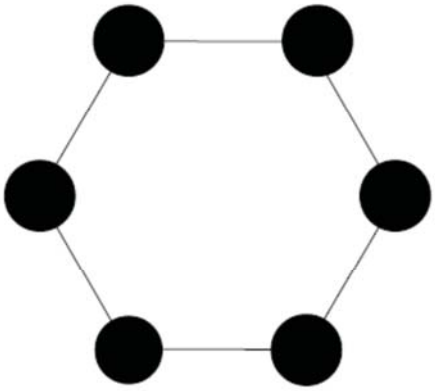

Figura 2.7 Algoritmo Paralelo de Grano Grueso [6].

#### 2.3.1.4 Algoritmos Híbridos

Algunas investigaciones han intentado combinar dos de los algoritmos mencionados anteriormente para paralelizar algoritmos genéticos, generando GAs paralelos híbridos. Cuando esto sucede se forma una jerarquía.

Las implementaciones jerárquicas pueden reducir el tiempo de ejecución más que cualquiera de sus componentes por si solas.

En la Figura 2.8 se muestra la posibilidad de combinar un algoritmo multipoblación en el nivel superior, con uno de grano fino en el nivel inferior.

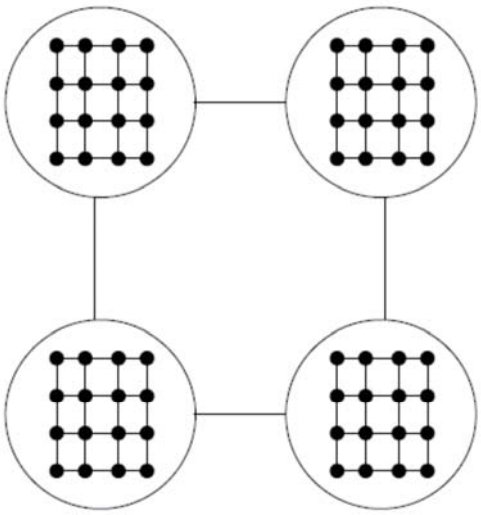

Figura 2.8 Representación de un algoritmo híbrido de tipo multipoblación en el nivel superior y de grano fino en el nivel inferior [6].

Otra posibilidad de combinación puede realizarse con un algoritmo multipoblación en el nivel superior y uno maestro-esclavo en el nivel inferior, como se muestra en la Figura 2.9.

Bianchini y Brown [9] presentaron un ejemplo de éste método, a través del cual consiguieron soluciones de la misma calidad de un GA paralelo maestro-esclavo o uno multipoblación, en menos tiempo.

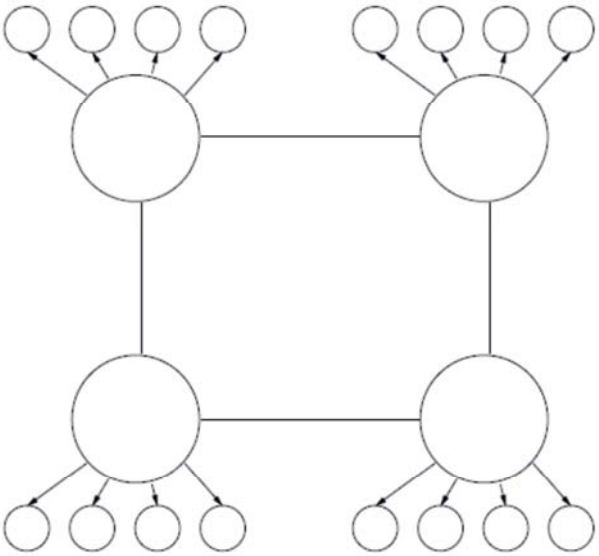

Figura 2.9 Representación de un algoritmo híbrido de tipo multipoblación en el nivel superior y maestro-esclavo en el nivel inferior [6].

Una tercera opción, mostrada en la Figura 10, es utilizar GA multipoblación en el nivel superior e inferior. Se puede conseguir una mejor distribución de las soluciones óptimas a través de las poblaciones, utilizando una topología altamente conectada en el nivel inferior y una frecuencia de migración elevada.

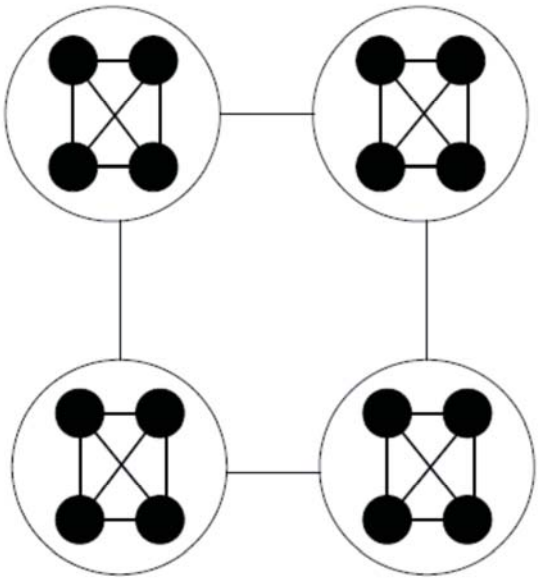

Figura 2.10 Representación de un algoritmo híbrido multipoblación en el nivel superior e inferior [6].

# 3 DARPTW

## 3.1 Dial-a-Ride Problem

En general, el Dial-a-Ride Problem (DARP) consiste en determinar el recorrido a realizar por uno o más automóviles que se encuentran dentro de una red de distribución, cumpliendo ciertas restricciones y requerimientos. Esto corresponde a sistemas de transporte de respuesta a la demanda.

Este problema va dirigido a transportar pasajeros, por lo que reducir las inconveniencias a los usuarios se debe equilibrar de alguna forma con la reducción de los costos de operación. En su ejecución se generan pedidos de transporte por parte de los usuarios, desde un lugar de origen específico llamado punto de recolección, hasta uno de destino llamado punto de entrega. El objetivo principal del problema radica en el diseño de un conjunto de rutas para los vehículos, cuyo costo sea mínimo y cumpla con todos los requerimientos de transporte realizados, además de satisfacer las distintas restricciones existentes, dentro de las cuales se pueden encontrar la capacidad, precedencia, ventanas de tiempo, tipo de servicio, entre otras [17].

## 3.2 DARPTW

DARPTW (Dial-a-Ride Problem with Times Windows) es un problema de optimización de múltiples objetivos, ya que existen dos factores críticos a ser optimizados: minimizar los costos totales de trasporte y además, maximizar el nivel de servicio que se ofrece a los usuarios, equivalente a minimizar el grado de insatisfacción que estos pudieran tener con el servicio requerido [18].

Existe un conjunto de peticiones de trasporte por parte de los usuarios, las cuales son conocidas previamente y no cambian durante la ejecución del algoritmo, lo que define este problema como estático. Cada petición determina una ventana de tiempo para la entrega del cliente hacia su lugar de destino y una ventana de tiempo para la recogida del cliente desde su lugar de origen. Se entrega el límite superior e inferior de la ventana de tiempo definida, considerando infactible una solución en la cual la entrega del cliente se realice en un tiempo fuera de la ventana, considerando así una ventana de tiempo rígida.

Para llevar a cabo el servicio, se dispone de una flota de vehículos homogéneos que poseen la misma capacidad de carga, la cual no puede ser sobrepasada. Los pasajeros son recogidos desde su lugar de origen y entregados en su lugar de destino por el mismo vehículo. De manera adicional, se dispone de múltiples depósitos para dichos vehículos, estos son locaciones particulares donde comienzan y terminan su recorrido. El depósito de término del recorrido no necesariamente debe ser el mismo que el de partida.

## 3.3 Optimización de múltiples objetivos

Un problema de optimización de múltiples objetivos, es aquel que involucra la optimización simultánea de más de una función objetivo, en el caso del DARPTW se requiere minimizar los costos de transporte y minimizar el descontento de los clientes respecto al nivel de servicio. Un problema de optimización de múltiples objetivos se puede expresar matemáticamente como [18]

$$Minimizar\ v(h) = \begin{bmatrix} v_1(h) \\ v_2(h) \\ \vdots \\ v_\sigma(h) \end{bmatrix}$$

donde $v_i(h)$, $i = 1,\ldots,n$ son las $n$ funciones objetivos en el problema. Este esquema es para funciones con los mismos parámetros, en otro caso se requiere modificar el criterio de la función objetivo para obtener resultados adecuados.

Este tipo de problemas se puede resolver de dos maneras:

- Estableciendo una escala de prioridades de los objetivos y resolviendo el problema en el orden de estas prioridades.
- Multiplicar una constante de "peso" con cada función objetivo y agruparlas en una suma.

Para DARPTW la segunda forma de resolución es la más conveniente, dado que es más práctico tener un control detallado, tanto de los factores del costo de operación como del nivel del servicio, que se traduce en captar mayor cantidad de clientes.

## 3.4 Calidad del Servicio

En [20] se menciona que existen dos tipos de ventanas de tiempo, las explícitas y las implícitas. Esta clasificación se relaciona directamente con la que se presentará más adelante en la sección 3.6. Las explicitas son las más comunes, corresponden con los datos de entrada al problema, las entrega el cliente y se deben respetar para que el servicio prestado sea válido o adecuado, según sea el caso. Las implícitas por otra parte, son más particulares, especialmente de sistemas de transporte como DARPTW. Surgen a causa de la necesidad de controlar las inconveniencias del cliente. Este generalmente proporciona una hora de entrega o de recogida esperada, lo que fija un extremo de una ventana. Como el cliente no quiere llegar tarde a su destino, el sistema que desarrolle el plan de transporte debe establecer el otro extremo de la ventana según corresponda. Con esto se tiene una ventana media abierta. Para prevenir que el transporte comience muy temprano respecto a su hora de entrega, o que la entrega se haga muy tarde respecto a su hora de recogida definida, se construye otra ventana de tiempo en la que el servicio se considera válido o adecuado. En la sección 3.7 se discute cómo se construyen estas ventanas implícitas.

Al tener ventanas de tiempo regulando la viabilidad y calidad de las soluciones, sin duda que la complejidad del problema se incrementa considerablemente. A diferencia de otros tipos de problemas de optimización, en DARPTW se trabaja con personas, por ello la calidad de servicio es un aspecto crítico a considerar como parte de los objetivos. Por ejemplo, en otros casos lo transportado son objetos. Más allá de la capacidad del vehículo, no es de mayor importancia el tiempo de viaje dentro de este, salvo excepciones puntuales, donde se transporten productos muy susceptibles al ambiente. En el caso de transporte de pasajeros, en cambio, las restricciones vinculadas al tiempo de viaje juegan un rol esencial.

## 3.5 Versiones del DARPTW

DARPTW presenta dos situaciones; una versión estática y una versión dinámica. En este contexto, estático o dinámico depende de la cantidad de información que se conoce de antemano al abordar el problema. En el caso en que toda la información sea conocida, se tratará de un problema estático. Basta con que parte de la información sea desconocida para que el problema se transforme en dinámico.

Los problemas estáticos son encontrados principalmente cuando se modela en un nivel de planificación táctico o estratégico. En estos casos los datos en el nivel operacional son construidos en base a información histórica o proyectada. El modelo es utilizado para revisar la gama de soluciones en distintos escenarios, usando procesos de simulación especiales. Para DARPTW, la información más relevante en este aspecto corresponde a las solicitudes de los clientes.

Los problemas dinámicos pueden ser formulados tras una serie discreta de problemas estáticos, donde el factor tiempo va impulsando la definición de nuevos problemas. Esta aproximación no suele ser la más efectiva, a veces es imposible de concretarse, por ello es que se utilizan otros mecanismos. Cuando se busca resolver modelos dinámicos reales y complejos, donde el tiempo computacional es crítico, es factible resolver primero el problema estático, y en el transcurso del tiempo y con el cambio de los datos, aplicarle correcciones con heurísticas de reconstrucción apropiadas. Para DARPTW, las rutas de los vehículos van construyéndose en tiempo real, acorde a nuevas peticiones de clientes, inconvenientes que se presenten, entre otras consideraciones.

Para el caso dinámico, los métodos de solución deben ser lo suficientemente rápidos para que un cliente que está realizando una solicitud pueda tener una respuesta concreta y en un tiempo razonable. En el caso estático esto no es tan importante considerando que las peticiones se tienen de antemano.

En este trabajo se aborda el caso estático de DARPTW.

## 3.6 Clasificación de Clientes

La clasificación de clientes se vincula directamente a las características de la solicitud del servicio que realizan los clientes. Un problema DARP genérico suele considerar ambos tipos [18].

Los clientes "de entrada" (*inbound customers*) son aquellos que están en alguna locación, generalmente vinculada al término de alguna jornada en su rutina. Ellos desean ser recogidos en esa locación, a alguna hora específica, para ser trasladados hasta algún otro lugar, usualmente sus hogares. Es aceptable que haya una pequeña demora desde la hora que ellos especifican hasta la hora en que efectivamente llega el transporte a recogerlos, sin embargo no corresponde que este último llegue antes de la hora especificada, pues el cliente no está preparado aún para partir. El cliente no define una ventana de tiempo explicita para la llegada a su destino.

Los clientes "de salida" (*outbound customers)* por otro lado son aquellas personas que requieren llegar a alguna locación, como su lugar de trabajo, de estudio, etc., a una hora específica. En este caso, el cliente puede ser recogido a cualquier hora, es decir, no define una ventana de tiempo de recogida explicita, pero el vehículo que lo transporte debe llegar al lugar de destino necesariamente antes de la hora que el cliente requiere.

Como se mencionó en la sección 4.4, aunque sólo se especifica una ventana de tiempo de parte del cliente, se deben definir ventanas de tiempo implícitas, según sea cliente de entrada o de salida. En el caso de este trabajo, se trabajará con ambos tipos de clientes, especificando tanto el límite superior como el inferior de la ventana de tiempo correspondiente.

## 3.7 Ventanas de Tiempo

Una ventana de tiempo corresponde a un intervalo de tiempo dentro del cual, del punto de vista de DARPTW, es válido o adecuado que un cliente sea recogido o entregado en una locación, según la operación que esté realizando en el momento el vehículo asociado. Que la situación sea válida o adecuada, dependerá del tipo de ventana de tiempo que se trate, podemos clasificarlas en ventanas de tiempo blandas y rígidas.

Cuando una ventana de tiempo es blanda (*soft time window*), es válido que un vehículo llegue fuera del tiempo que corresponde a la ventana. Esto implica que la solución pertinente no será considerada como inválida. Sin embargo, cuando se usa este tipo de ventanas, se suele agregar en términos de calidad de servicio un factor de influencia importante en la función objetivo. De esta forma, si bien la solución no será desechada, su calidad bajará en la medida que se no se respeten las ventanas de tiempo.

En otro caso, una ventana de tiempo es rígida (*hard time window*) cuando su incumplimiento implica directamente que la solución asociada es inválida.

La definición de las ventanas de tiempo, es decir, sus rangos inferiores y superiores, depende del tipo de cliente que se considere para el problema y otras variables presentes. En lo que sigue, se considera a *i* como una solicitud de transporte particular, esto es, un cliente puntual.

### 3.7.1 Clientes de Entrada

En este caso las ventanas son establecidas en base al horario de recogida establecido por el cliente. Este tiempo $a_i$ corresponderá al rango inferior de la ventana de tiempo de recogida del cliente. El rango superior de esta última, $b_i$, si no es especificado, se puede obtener con un tiempo límite *dev* establecido en el sistema, como una desviación máxima para la espera del cliente en la recogida, de esta forma, $b_i = a_i + dev$.

Para obtener el límite inferior de la ventana de tiempo de entrega $a_{n+i}$, se requiere el tiempo más temprano en el que el cliente podría ser entregado a su lugar de destino, esto equivale al rango inferior de la ventana de recogida $a_i$, más el tiempo directo entre el punto de recogida y el punto de entrega, lo que se conoce como tiempo directo de viaje (*direct ride time* o *DRT*), y el tiempo de servicio del cliente *i*, $s_i$. De esta forma $a_{n+i} = a_i + s_i + DRT_{i,n+i}$.

El límite superior de la ventana de tiempo de entrega $b_{n+i}$, requiere de un parámetro adicional, que en la literatura suele encontrarse de dos formas. Por un lado, podemos considerar el máximo exceso en el tiempo de viaje *E* (*maximum excess ride time)*, el cual se puede obtener a través de una ecuación lineal simple dependiente del *DRT*, por ejemplo $E_{i,n+i} = 5\ min. + 0.5\ DRT_{i,n+i}$. Este valor es sumado a $b_i$ y al tiempo de servicio para obtener el límite superior de la ventana de tiempo de entrega, esto es, $b_{n+i} = b_i + s_i + DRT_{i,n+i} + E_{i,n+i}$. Por otro lado, está el tiempo máximo de viaje *MRT* (*maximum ride time*) que es igual al anterior, pero incluye implícitamente a *DRT*. Para cumplir con esto, en la ecuación lineal *DRT* es multiplicado por un número mayor o igual que 1, por ejemplo $MRT_{i,n+i} = 8\ min. + 2\ DRT_{i,n+i}$. De esta forma, el límite superior de la ventana de tiempo de entrega sería $b_{n+i} = b_i + s_i + MRT_{i,n+i}$.

En la Figura 3.1 se muestra la construcción de las ventanas de tiempo en este caso, considerando el uso de *MRT* para el cálculo de límite superior de la ventana de entrega.

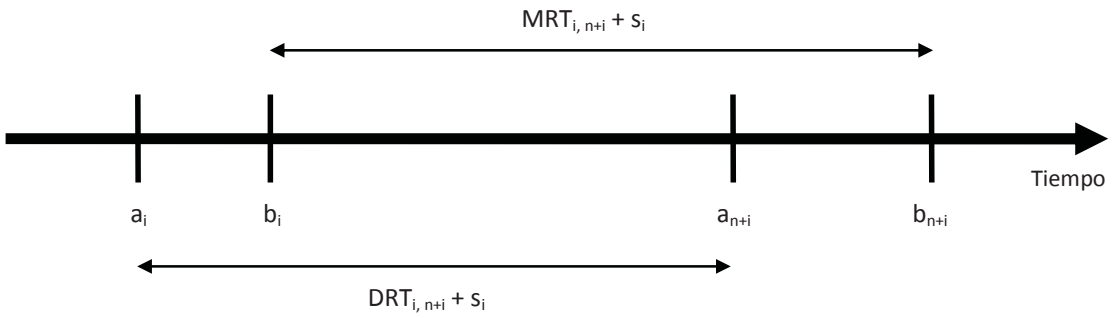


Figura 3.1 Construcción de las ventanas de tiempo para clientes de entrada [18].

### 3.7.2 Clientes de Salida

Este caso es análogo al anterior, considerando que los límites se van obteniendo hacia atrás en el tiempo. Acá las ventanas son establecidas en base al horario de entrega establecido por el cliente. Este tiempo $b_{n+i}$ corresponderá al rango superior de la ventana de tiempo de recogida del cliente. El rango inferior de esta última, $a_{n+i}$, si no es especificado, se puede obtener del valor *dev*, de esta forma, $a_{n+i} = b_{n+i}$ - *dev*.

Para obtener el límite superior de la ventana de tiempo de recogida $b_i$, se resta *DRT* y el tiempo de servicio a $b_{n+i}$, esto es $b_i = b_{n+i} - (s_i + DRT_{i,n+i})$. El límite inferior de la ventana de tiempo de recogida $a_i$, usando tanto *E* como *MRT*, resultar como $a_i = a_{n+i} - (s_i + DRT_{i,n+i} + E_{i,n+i})$ y $a_i = a_{n+i} - (s_i + MRT_{i,n+i})$ respectivamente.

En la Figura 3.2 se muestra la construcción de las ventanas de tiempo en este caso, considerando el uso de *MRT* para el cálculo de límite superior de la ventana de recogida.

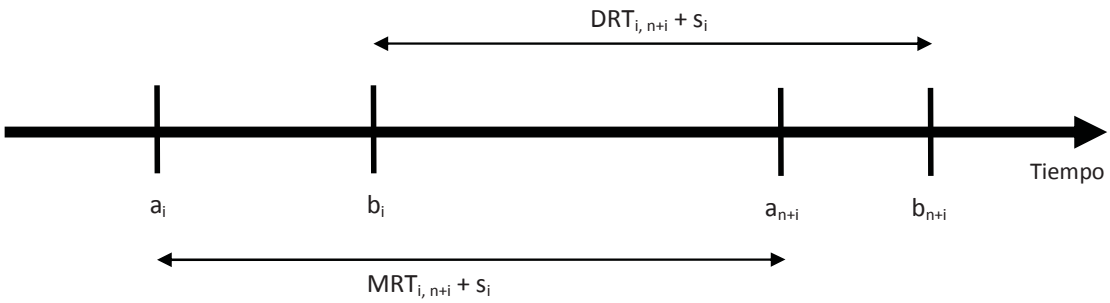


Figura 3.2 Construcción de las ventanas de tiempo para clientes de salida [18].

En esta investigación se consideró la construcción de ventanas, usando el factor *MRT* y con ventanas rígidas.

## 3.8 Rutas

Una consideración adicional en términos de las características de DARPTW, corresponde a la planificación del recorrido de un vehículo. Este aspecto puede influir directamente en consideraciones de implementación de un sistema, por ejemplo, en lo que es la función objetivo. A continuación se presentan dos aproximaciones al respecto.

Por una parte, la ruta que desarrolla un vehículo puede estar conformada por bloques de planificación. Un esquema de estos bloques se muestra en la Figura 3.3. Estos bloques se

caracterizan por contener eventos tanto de recogida como de entrega de un conjunto de clientes en la ruta. Los bloques en sus extremos pueden tener depósitos de vehículos y tiempos de inactividad conocidos como *slacks*. En este contexto, se dice que un vehículo puede estar en tres estados distintos: En uno, está en un depósito, por lo que no ha comenzado su recorrido, o ya lo completó. El otro estado es de actividad, en el que se encuentra viajando para recoger o entregar pasajeros. Un último estado es de inactividad, en donde el vehículo está detenido, esperando un horario en que volverá a estar activo para servir pasajeros pertenecientes a un bloque posterior. En este contexto, un vehículo jamás estará inactivo con pasajeros a bordo, pues los *slacks* se ubican entre bloques.

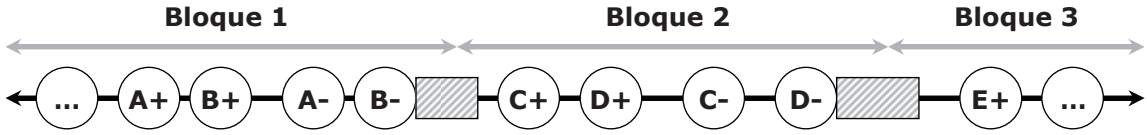


Figura 3.3 Estructura de un bloque de planificación [18].

Otra aproximación ya no considera de forma tan directa los bloques. Se permiten tiempos de inactividad aún con pasajeros a bordo. Esta forma de abordar las rutas es menos restrictiva que la anterior, pero tiene un costo importante en términos de calidad de servicio, que debe ser considerado en la función objetivo. Este costo es mesurado según el tiempo que los pasajeros permanecen dentro de un vehículo detenido.

Los tiempos de inactividad surgen cuando en una ruta, el viaje inmediato desde un evento origen a un evento destino implica que se llegue muy temprano a este último. Si bien estos *slacks* pueden parecer negativos desde el punto de vista del problema, su existencia agrega cierto grado de flexibilidad a las soluciones, y los *slacks* a su vez pueden ser útiles cuando se quiere reconstruir una ruta, por ejemplo con la inserción de un nuevo pasajero en ella.

En el contexto de este trabajo, se considerará la primera aproximación, en la que los tiempos de inactividad pueden existir sólo con los vehículos vacíos.

## 3.9 Modelo matemático

A continuación se muestra el modelo matemático de DARPTW, extraído de [18]. Este modelo trabaja con ventanas de tiempo rígidas, lo que se aproxima a las necesidades que se consideran en este trabajo.

El modelo se basa en los siguientes datos:

- Se tienen $n$ de solicitudes de servicio. Para una petición, $i$ es el punto de recogida y $n+i$ corresponde al punto de entrega. $d_i$ corresponde a la cantidad de pasajeros a transportar en la solicitud $i$.
- $P = \{1,\ldots, n\}$ y $D = \{n+1,\ldots, 2n\}$ son el conjunto de puntos de recogida y de entrega respectivamente.
- $N = \{P \cup D\}$ es el conjunto total de puntos especificados en las solicitudes.
- Para cada vehículo $k \in K$, se tienen dos depósitos, el de origen $o(k)$ y el de destino $d(k)$.
- $A = N \cup \{o(k), d(k)\}\ \forall k \in K$ es el conjunto total de puntos del problema.
- $V \subset K$ es el conjunto de los vehículos usados efectivamente en la solución y $v$ es su cantidad.
- $C^k$ corresponde a la capacidad máxima del vehículo $k$.
- $l_i = d_i$ corresponde a la diferencia de carga en el nodo $i$ y $l_{n+i} = -d_i$ corresponde a la diferencia de carga en el nodo $n+i$.
- $L_i^k$ corresponde a la carga del vehículo $k$ después de visitar el nodo $i$.
- Cada nodo $i$ debe ser servido en la ventana de tiempo $[a_i, b_i]$.
- $T_i^k$ corresponde al tiempo inicial de servicio del vehículo $k$ en el nodo $i$.
- $t_{i,j}$ corresponde al tiempo de conducción entre el nodo $i$ y el nodo $j$.
- $s_i$ corresponde al tiempo de servicio en el nodo $i$.
- $x^k_{i,j}$ es una variable de decisión con valor 1 si el vehículo $k$ sirve al nodo $i$ y después conduce para servir al nodo $j$. En caso contrario, la variable es 0.

Considerando la técnica de multiplicar constantes mencionada en 3.3 (en este caso $\alpha$, $\beta$ y $\gamma$), se define la siguiente función objetivo:

$$\min\ \alpha \sum_{k\epsilon V} \sum_{(i,j)\epsilon A} t_{i,j} x_{i,j}^k + \beta v + \gamma \sum_{k\epsilon V} \sum_{i\epsilon P} (T_{n+i}^k - s_i - T_i^k) \tag{3.9.1}$$

Como se ve, son tres los elementos que se evalúan en esta función. La parte que esta multiplicada por $\alpha$ corresponde al factor del tiempo de viaje de todos los vehículos en la solución. La parte multiplicada por $\beta$ es simplemente la cantidad de vehículos usados en la solución. Los dos factores anteriores están enfocados a los costos de transporte en sí. El tercer elemento, multiplicado por $\gamma$ corresponde al tiempo de viaje total de los pasajeros sobre los vehículos. Naturalmente, este es el factor vinculado a la calidad de servicio.

Además, la función objetivo está sujeta a las siguientes restricciones:

$$\sum_{k\epsilon V}\sum_{j\epsilon P\cup d(k)} x^k_{o(k),j} = v \tag{3.9.2}$$

$$\sum_{k\epsilon V}\sum_{i\epsilon D\cup o(k)} x^k_{i,d(k)} = v \tag{3.9.3}$$

$$\sum_{k\epsilon V}\sum_{j\epsilon A} x^k_{i,j} = 1, \forall i \epsilon N \tag{3.9.4}$$

$$\sum_{j\epsilon N} x^k_{i,j} - \sum_{j\epsilon N} x^k_{j,n+i} = 0, \forall k \epsilon V, i \epsilon P \tag{3.9.5}$$

$$x^k_{i,j}\left(T^k_i + s_i + t_{i,j} - T^k_j\right) \leq 0, \forall k \epsilon V, (i,j) \epsilon A \tag{3.9.6}$$

$$a_i \leq T^k_i \leq b_i, \forall k \epsilon V, i \epsilon A \tag{3.9.7}$$

$$T^k_i + s_i + t_{i,n+i} \leq T^k_{i+n}, \forall k \epsilon V, i \epsilon P \tag{3.9.8}$$

$$x^k_{i,j}\left(L^k_i + l_j - L^k_j\right) = 0, \forall k \epsilon V, (i,j) \epsilon A \tag{3.9.9}$$

$$l_i \leq L^k_i \leq C^k, \forall k \epsilon V, i \epsilon P \tag{3.9.10}$$

$$L^k_{o(k)} = L^k_{d(k)} = 0, \forall k \epsilon V \tag{3.9.11}$$

$$x^k_{i,j} \epsilon \{0,1\}, \forall k \epsilon V, (i,j) \epsilon A \tag{3.9.12}$$

(3.9.2) y (3.9.3) obligan a que el número de vehículos que abandona un depósito y el número de vehículos que llega a este sea igual al número de vehículos considerados en la solución.

(3.9.4) y (3.9.5) son validaciones referentes a los clientes que son servidos por los vehículos. En (3.9.4) se asegura que todas las solicitudes de los clientes del problema son atendidas y en (3.9.5) se establece que cuando un cliente es recogido por un vehículo, debe ser entregado por el mismo.

(3.9.6) es una validación respecto a las rutas que considera la solución. Para cada uno de los desplazamientos desde un nodo a otro en la solución, el tiempo de salida de un nodo de origen debe ser menor al tiempo de llegada al nodo destino.

(3.9.7) es la validación referente a las ventanas de tiempo.

(3.9.8) es igual que (3.9.6), pero en el contexto de cualquier par de nodos que conforme una solicitud de servicios de parte de un cliente.

(3.9.9) asegura que entre dos eventos de una ruta solución, la carga del vehículo sea correctamente calculada.

(3.9.10) asegura que la capacidad de un vehículo nunca sea superada.

(3.9.11) es una validación respecto a la carga inicial y final de cada vehículo en su recorrido, las que deben ser 0.

(3.9.12) son las variables usadas en el modelo.

## 3.10 Problemas relacionados

Algunos problemas relacionados se describen brevemente a continuación.

### 3.10.1 Problema del vendedor viajero (TSP-Traveling Salesman Problem)

En general, TSP [41] consiste en determinar cómo se debe recorrer la totalidad de un conjunto de puntos, sin pasar dos veces por un mismo lugar, volviendo al punto desde donde se partió, minimizando el camino total recorrido.

En teoría de grafos, este problema consiste en que dado un grafo de N vértices (correspondientes a las ciudades) se debe encontrar el camino o ruta más corta, que partiendo desde una ciudad origen, pase por todas las ciudades y lo haga sólo una vez. La manera de evaluar cuál será el camino más corto, será a través de las aristas existentes entre los vértices, las que tendrán un valor que indicará la longitud (peso) entre una ciudad y otra.

La formulación para este problema es la siguiente: un individuo tiene que visitar $n-1$ ciudades partiendo de una ciudad inicial (ciudad 1) y volver a ella, de forma que la ruta elegida sea la más corta o la más barata. Para lo anterior existen $(n-1)!$ soluciones factibles, llamadas rutas.

En la Figura 3.4 se observan algunas variaciones de este problema:

- On-line Travelling Salesman Problem: Se considera un modelo “online” en el cual las peticiones de servicio son conocidas en el tiempo, esto es mientras el vendedor va atendiendo requerimientos previos.
  Se han estudiado dos versiones del problema. Uno es Nomadic On-line Travelling Salesman Problem (NOLTSP), que considera el tiempo de finalización requerido para atender todas las solicitudes presentadas, y otro es Homing Online Travelling Salesman Problem (HOLTSP), cuyo objetivo es minimizar el tiempo de finalización requerido para servir todas las peticiones presentadas y retornar al origen.
- TSP con ventanas de tiempo (TSPTW): al igual que el problema tratado en esta en esta investigación, existe una variación con ventanas de tiempo. En este caso, cada ciudad lleva asociada un intervalo de tiempo en que tiene que ser visitada.

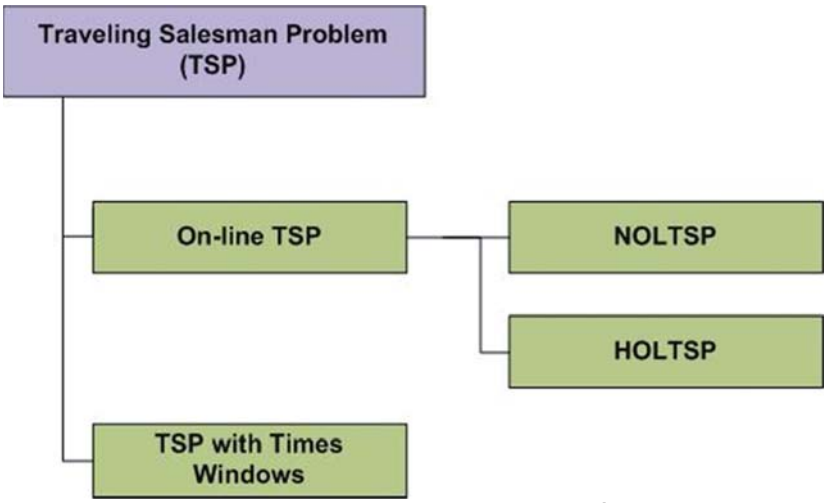

Figura 3.4 TSP y algunas de sus versiones [Elaboración propia].

### 3.10.2 Problema de rutas vehiculares (VRP- Vehicle Routing Problem)

El objetivo de este problema es atender un conjunto de clientes con una demanda por servicio a un costo mínimo, a través de rutas que tienen un origen y un término en un depósito central. Se considera una flota de vehículos ubicados en el depósito central, los cuales proveen un servicio a clientes que se encuentran geográficamente dispersos. Por lo anterior, se considera el diseño de rutas para atender las peticiones (ya sean de distribución o recolección) desde uno o más depósitos centrales a un conjunto de clientes dispersos.

El VRP tiene un planteamiento basado en el TSP, por lo que adopta sus técnicas de solución, utilizando algoritmos de éste incorporando algunas variaciones.

Se puede formular el problema de la siguiente manera: un número $n-1$ de clientes en ciudades concretas con demandas conocidas de cierto artículo, han de ser atendidos desde una central (ciudad 1) por un número de vehículos $m$ de capacidad conocida. Se trata de diseñar rutas de vehículos con distancia total a recorrer mínima.

En la Figura 3.5 se observan algunas versiones de este problema, que varían dependiendo de ciertos factores, restricciones u objetivos.

- VRP con capacidad limitada (CVRP- Capacited VRP): es el VRP general, donde existen uno o varios vehículos con capacidad limitada, encargados de satisfacer peticiones según la demanda de los clientes.
- VRP con múltiples depósitos (MDVRP- Multi-Depot VRP): cada depósito (con su flota de vehículos) debe servir a todos los clientes.
- Period VRP (PVRP): su planteamiento contempla un horizonte de operación de $m$ días. Dentro de este periodo el cliente debe ser visitado una vez.
- VRP de entrega dividida (SDVRP- Split Delivery VRP): se considera una entrega dividida, en la cual se permite que un cliente pueda ser atendido por

varios vehículos si debido a esto el costo total se reduce. Esto es importante si el tamaño de los pedidos excede la capacidad de un vehículo.

- Stochastic VRP (SVRP): Contempla uno o varios componentes aleatorios; clientes, demanda y tiempos pueden ser estocásticos.
- VRP con entrega y recogida (VRPPD- VRP with Pickup and Delivery): en esta versión existe la posibilidad de que los clientes puedan devolver determinados bienes, por lo cual se debe considerar que el vehículo tenga capacidad para ambas acciones.
- Mix Fleet VRP (MFVRP): se consideran vehículos con distintas capacidades. Por lo anterior, estas capacidades se deben tomar en cuenta en la ruta que seguirá cada recurso, ya que un vehículo con mayor capacidad puede cubrir una ruta en que la demanda sea mayor.
- VRP con ventanas de tiempo (VRPTW- VRP with Time Windows): la restricción adicional en este caso, corresponde a una ventana de tiempo en cada cliente. Lo anterior se refiere a que cada cliente sólo está dispuesto a recibir la visita del vehículo durante un intervalo de tiempo dado.

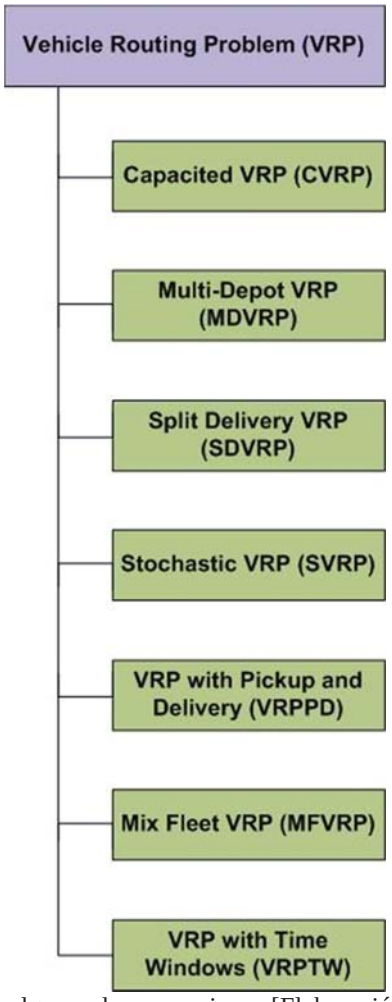


Figura 3.5 VRP y algunas de sus versiones [Elaboración propia].

## 3.11 Computación Paralela y Optimización de Múltiples Objetivos

Los sistemas de computación paralelos han sido utilizados ampliamente en el campo de la optimización monobjetivo [21], [22], [23]. En el caso de las técnicas deterministas, un ejemplo típico consiste en la resolución de problemas de optimización mediante algoritmos paralelos de Ramificación y Poda [24]. Se trata, en general, de resolver problemas en menos tiempo o bien resolver problemas más complejos. En el contexto de los métodos heurísticos, el paralelismo no solo significa resolver los problemas de forma más rápida, sino que además se obtienen modelos de búsqueda más eficientes: un algoritmo heurístico paralelo puede ser más efectivo que uno secuencial, aun ejecutándose en un solo procesador. Sirvan como ejemplo los estudios sobre algoritmos evolutivos presentados en [25], [26].

Las ventajas que ofrece la programación paralela a la optimización monobjetivo también se mantienen en la optimización multiobjetivo.

Algunos estudios referentes a técnicas evolutivas son [27], [28], [29]. Sin embargo, como se afirma en [27], los trabajos sobre implementaciones paralelas han sido escasos en este campo.

En general, la mayoría de los trabajos sobre computación paralela y optimización están centrados en dos tipos de sistemas paralelos: los sistemas multiprocesadores de memoria compartida y los sistemas distribuidos, estos últimos basados en clusters o redes de área local. En el primer caso, los servicios ofrecidos por el sistema operativo son suficientes para desarrollar programas paralelos: se pueden usar lenguajes secuenciales con bibliotecas de hebras o llamadas al sistema para escribir aplicaciones multi-hebra o multi-proceso que utilicen los distintos procesadores del sistema. Otra opción consiste en usar lenguajes paralelos como Java.

En el caso de los sistemas distribuidos, existe una cantidad enorme de aspectos a considerar, como la red de conexión (Ethernet, Myrinet, ATM), las distintas topologías (anillo, árbol, hipercubo, malla), los modelos de programación (paso de mensajes, RPC, memoria compartida distribuida), las bibliotecas de comunicación (MPI, PVM, sockets), lenguajes paralelos (Java, C#), middleware (CORBA, DCOM, RMI) e incluso tecnologías de Internet (XML, SOAP, Servicios Web).

La última generación de sistemas distribuidos está basada precisamente en la popularidad de Internet y la disponibilidad de una gran cantidad de recursos computacionales que están repartidos geográficamente. Estos recursos pueden agruparse para ofrecer un recurso computacional único y unificado. Este proceso ha desembocado en lo que se denomina como Grid computing [30], aunque también se usan términos como metacomputing, Internet computing o Web computing.

La idea de utilizar cientos o miles de procesadores es muy atractiva, ya que puede permitir la resolución de problemas de optimización considerados como intratables. Por ejemplo, en [31], se resuelve, en una semana, una instancia del problema de asignación cuadrática (QAP, Quadratic Assignment Problem) que habría necesitado 7 años de tiempo de

cómputo en una estación de trabajo, utilizando para ello una red computacional con unos 2500 procesadores. El problema se resolvió con un algoritmo distribuido de Ramificación y Poda. Con excepción de este trabajo, las referencias relacionadas con optimización y computación Grid que se encuentran son escasas [32], [33], [34].

En lo que al resultado de esta investigación se refiere, no existe trabajo alguno relacionado con algoritmos evolutivos para optimización multiobjetivo y sistemas multiagente; por tanto, este trabajo supone la primera tentativa en este sentido.

## 3.12 Estudio de Algoritmos Genéticos para el Problema de Transporte de Pasajeros (DARPTW)

En esta sección se tratarán los principales puntos de la investigación en la que fue basado el presente trabajo. Este es un estudio sobre la implementación de Algoritmos Genéticos para el Problema de Transporte de Pasajeros con Ventanas de Tiempo, tratado por Urra E. en [18].

Se desarrollaron dos modelos (ver Figura 3.6 y 3.7). Cada uno de ellos considera ciertos elementos que pueden ser divididos en dos grupos: submodelos de soporte, que se utilizan para el manejo de restricciones y factibilidad (esquemas de factibilidad preliminar, mecanismos de modificación de rutas, generación de soluciones factibles) y submodelos de GA, que corresponden a los elementos a considerar dentro de la formalización de un GA (genotipo, fenotipo, selección, recombinación, mutación, población inicial).

En este trabajo se utilizará sólo el modelo LLGA (ver Figura 3.6), ya que fue el que entregó mejores soluciones en la investigación que se describe a continuación.

### 3.12.1 Esquemas de factibilidad preliminar

#### 3.12.1.1 Tabla de precedencia preliminar en ejecución de eventos

Es posible indagar en los datos de entrada para establecer relaciones que permitan acotar el espacio de búsqueda y hacer el procesamiento más rápido. En este caso se considera una tabla de precedencia preliminar en ejecución de eventos y, en base a ella, la generación de una tabla de clientes incompatibles.

La forma en que se establece la precedencia dependerá de las suposiciones que se hagan, pero existen ciertas relaciones lógicas entre eventos que tienen un esquema de precedencia dado. Sea la expresión X => Y equivalente a “X precede preliminarmente a Y”, y X $\not\Rightarrow$ Y equivalente a “X no precede preliminarmente a Y”, entonces se puede considerar lo siguiente:

- Si (X => Y) $\wedge$ (Y =>X), entonces X e Y tienen una relación de dependencia cíclica.
- Si (X $\not\Rightarrow$ Y) $\wedge$ (Y $\not\Rightarrow$ X), entonces X e Y son preliminarmente paralelos.

- Si (X => Y) $\wedge$ (Y => Z), entonces X => Z, si es que ambos eventos están en la misma ruta de Y.

La dependencia cíclica implica la incompatibilidad de la existencia de eventos en una misma ruta. Por otro lado, la precedencia preliminar sirve para evitar los casos infactibles, pero no para determinar los casos factibles, esta es tarea de otros elementos de este marco de trabajo.

#### 3.12.1.2 Incompatibilidad de clientes

La dependencia cíclica entre eventos provee una forma efectiva de identificar condiciones de infactibilidad, no sólo a nivel de rutas sino también a nivel de asignación de clientes a vehículos o clusters.

Por ejemplo, si se da el caso $(B+=>C+) \wedge (C+=>B+)$, se estaría en presencia de una dependencia cíclica, la cual permite definir directamente la imposibilidad de colocar a dichos eventos en una misma ruta. Al mismo tiempo, esto implica que los clientes asociados a dichos eventos, B y C, nunca podrán asignarse al mismo vehículo de forma factible. De este modo se asume que B y C son clientes incompatibles.

La incompatibilidad de clientes provee un importante esquema de factibilidad por medio del cual se pueden generar más fácilmente asignaciones de clientes a vehículos. Esto es especialmente práctico en la generación de la población inicial del GA.

### 3.12.2 Mecanismos de modificación de rutas

En esta parte, se evalúa la modificación de una ruta en base a la inserción, eliminación o el intercambio de un par de eventos en ella, considerando todas las restricciones importantes: ventanas de tiempo y carga de los vehículos. Para la inserción y eliminación, se consideran los eventos asociados a un cliente, mientras que en el intercambio pueden ser eventos de clientes distintos. La factibilidad en las operaciones de inserción e intercambio se evalúa en parte, usando la tabla de precedencia preliminar.

#### 3.12.2.1 Heurística propuesta

Se consideró una modificación de la heurística de inserción factible descrita en el Framework MADARP propuesto en [38]. Aquí se definen una serie de valores precalculados para los eventos que permiten evaluar qué tanto pueden ser anticipado o postergado el tiempo de llegada del vehículo al evento, con el fin de evaluar una posible inserción.

Una desventaja que tiene la heurística descrita anteriormente, es que no considera la eventual modificación de los bloques tras una inserción. La nueva heurística se basa en la idea de reconstruir la ruta modificada, desde el punto en que empiezan los cambios, hasta el último punto que puede ser afectado por ellos en la ruta, que no necesariamente corresponde al fin del bloque. La idea es empezar a realizar intersecciones de ventanas de tiempo a medida que se avanza en la ruta. Las ventanas que se obtienen son las siguientes:

- Ventana base (*[ET, LT]*): ventana de tiempo que se entrega como dato de entrada en relación a la solicitud de transporte asociada al evento.
- Ventana acumulada hacia adelante (*[BET, BLT]*): resultado de intersectar la ventana acumulada hacia adelante del evento anterior en la ruta con la ventana base del evento actual.
- Ventana factible relativa al evento (*[FET, FLT]*): resultado final de todas las intersecciones, cuando se llega al último evento de la ruta, o al final de un bloque, relativo al evento actual.

#### 3.12.2.2 Inserción, eliminación e intercambio de pares

En la inserción de eventos, siempre se generará un segmento modificado, que puede contener como mínimo dos eventos (los que se insertarán) y como máximo todos los eventos en la ruta desde el punto de inserción hasta el final de la misma, incluyendo a los eventos que se insertarán.

La eliminación es más simple, ya que no se deben encontrar intervalos. Basta con recorrer la ruta y encontrar los eventos que se desean eliminar.

En el intercambio, se cambia el orden de un par de eventos contiguos en la ruta. El segmento modificado puede contener como mínimo un par de eventos, los que se intercambian, y como máximo todos los eventos de la ruta desde el primero que se intercambia hasta el final. Para evaluar su se evalúa si el primero precede preliminarmente al segundo. Si no es así, los eventos se asumen como intercambiables, dado que al principio el segundo ya está después que el primero, por ende no lo precede preliminarmente.

### 3.12.3 Generación de soluciones factibles

Se entregan soluciones genéricas para ser aplicadas a los genotipos de los modelos a usar en el algoritmo. El proceso consta de dos etapas:

- Generación de solución factible base: se tratan las dependencias cíclicas, evitando que los clientes incompatibles se asignen a las mismas rutas.
- Generación de solución factible final: con la solución factible base generada, se tiene cierta cantidad de clientes que aún no se han insertado en las rutas. Los mismos se van tomando y se van insertando en las rutas de la solución de forma aleatoria. Si al tratar de insertar en alguna ruta se produce algún error de factibilidad, se prueba con otra ruta distinta en la solución.

### 3.12.4 Representación o Genotipo

El genotipo juega un rol fundamental, pues es uno de los elementos que tiene mayor interacción o genera mayor dependencia respecto a otros elementos, muy en particular con lo que es la recombinación. Además, en él se pueden ver afectadas una serie de restricciones del problema.

#### 3.12.4.1 Representación tipo LLGA (Linkage Learning GA)

Considera qué pasajeros se asignan a qué vehículos, lo que se define como tipo clustering. Con esto surge una restricción respecto a que un pasajero puede ser asignado sólo a un vehículo y a la vez, este vehículo debe ser el que ejecute los servicios de recogida y entrega de dicho pasajero.

En la representación general del LLGA, un gen está compuesto de un par (locus, alelo), lo que permite ordenar de distintas maneras los elementos del cromosoma, en lugar de realizar un ordenamiento rígido, y obtener una solución idéntica gracias a la incorporación del locus.

En este caso, un gen estaría compuesto por el locus, el pasajero a asignar, y el vehículo al que se le asigna. El problema de esto último es que habría que realizar validaciones respecto a que en un cromosoma estén presente no sólo todos los locus, si no también todos los pasajeros. Por ello es que, considerando que el genotipo se enfocará a abordar sólo la asignación de pasajeros a vehículos, el locus se considerará equivalente al identificador del pasajero.

#### 3.12.4.2 Representación basada en rutas

Se enfoca en exponer explícitamente qué rutas tiene considerada la solución, abordando al mismo tiempo la asignación.

La codificación de un cromosoma en este genotipo corresponde a una lista de secuencias de eventos, donde cada secuencia corresponde a una lista de eventos doblemente enlazada y hace referencia a una ruta que se ejecuta por un vehículo, definiendo explícitamente el enrutamiento asociado. La cantidad de elementos en dicha lista indica la cantidad de vehículos considerados en la solución.

### 3.12.5 Recombinación

Se consideraron dos operadores de recombinación, cada uno relacionado directamente con las representaciones descritas anteriormente.

- Recombinación tipo LLGA: se inserta un segmento de uno de los padres en un punto específico del otro, eliminando los elementos repetidos fuera del segmento donado. Ya que se podrían llegar a pasar a llevar ciertas restricciones, se adopta una estrategia de verificación previa, donde se evalúa el segmento del cromosoma que se planea donar. Si para alguno de los genes la recombinación es infactible, se iniciará nuevamente la verificación en base a la selección de un nuevo segmento desde el cromosoma donante.
- Recombinación basada en rutas: en este tipo de recombinación se tienen dos padres para generar un hijo, uno es el padre “principal” y el otro es el “secundario”. Se ejecuta en las tres siguientes etapas:
    - Donación directa de rutas: el padre principal dona una parte menor de sus rutas al hijo de forma directa (se copian en éste).

- Donación filtrada de rutas: se donan rutas desde el padre secundario y se filtran a través de la eliminación de los clientes que ya se encuentran actualmente en el hijo, encogiendo en orden las que tienen mayor cantidad de clientes para ser traspasadas al hijo. La cantidad de rutas a donar dependerá del límite de vehículos del problema.
- Reparación por inserción: esta etapa se realiza sólo si faltaron clientes por insertar en la solución. Se construye una lista de los clientes que faltan y se van insertando de manera aleatoria en las rutas que ya tiene la solución.

Si la recombinación no es factible se vuelve a ejecutar el proceso.

### 3.12.6 Evaluación o Fenotipo y Selección

La función de evaluación que se utiliza para el DARPTW es la función objetivo del modelo que se presenta en. En general se evalúan 5 elementos: tiempo total de viaje de los vehículos, tiempo de duración de los slacks, cantidad de vehículos, exceso en el tiempo de viaje para los clientes (diferencia entre el tiempo de viaje directo desde punto de recogida a punto de entrega y el tiempo de viaje realizado en la ruta para llegar de un punto a otro) y el tiempo de espera de los clientes (diferencia entre el tiempo actual de llegada a ejecutar un servicio y el extremo inferior de la ventana de tiempo establecida para ese servicio), a cada uno de los cuales se le asigna una constante de peso.

En el caso de la cantidad de vehículos, depende mucho de los datos de entrada usados, dado que algunos definen como limitante la cantidad de vehículos a utilizar, por ende este factor no sería considerado.

Por otro lado, la selección permite la aplicación del resultado de una evaluación. Para los dos modelos descritos se utiliza la selección por torneo descrita en la sección 2.2.4. Las ventajas de este tipo de selección son la posibilidad de manejar la presión de la selección a través del tamaño del torneo y que el mapeo entre la prioridad de selección de los integrantes del torneo es directa: los que tienen una función de evaluación con resultado menor, son los que tienen mayores posibilidades de ganar el torneo.

### 3.12.7 Mutación

Son considerados dos tipos de mutación, ambos de tratan de manera independiente, cada uno con su propia probabilidad de ocurrencia.

- Mutación de clusters: se cambia el vehículo de uno de los pasajeros en el cromosoma. Cada vez que esto sucede, se realiza una verificación correspondiente basada en la eliminación del cliente desde el vehículo de origen y la inserción del mismo en el vehículo destino y considerando la incompatibilidad entre clientes. Si el resultado es negativo, se intenta con otro vehículo y en caso de que no se pueda cambiar ningún vehículo la mutación se anula. Por cada cliente se evalúa la probabilidad de realizar el cambio de vehículo, por ello dicha probabilidad no puede ser muy grande.
- Mutación de rutas: se intercambian eventos que tienen posibilidad de cambiar de orden. Se evalúa la probabilidad para cada ruta en el problema, se selecciona un punto

"intercambiable" de la ruta de forma aleatoria y se evalúa el intercambio. Si este es infactible, se intenta con otro punto. En caso de no poder intercambiar ninguno de los eventos en la ruta, se anula la mutación.

### 3.12.8 Población inicial

Para la población inicial se utiliza la herramienta de generación de soluciones factibles explicada en la sección 3.12.3. La forma en que estas soluciones se incorporan a los cromosomas, dependerá de cada genotipo.

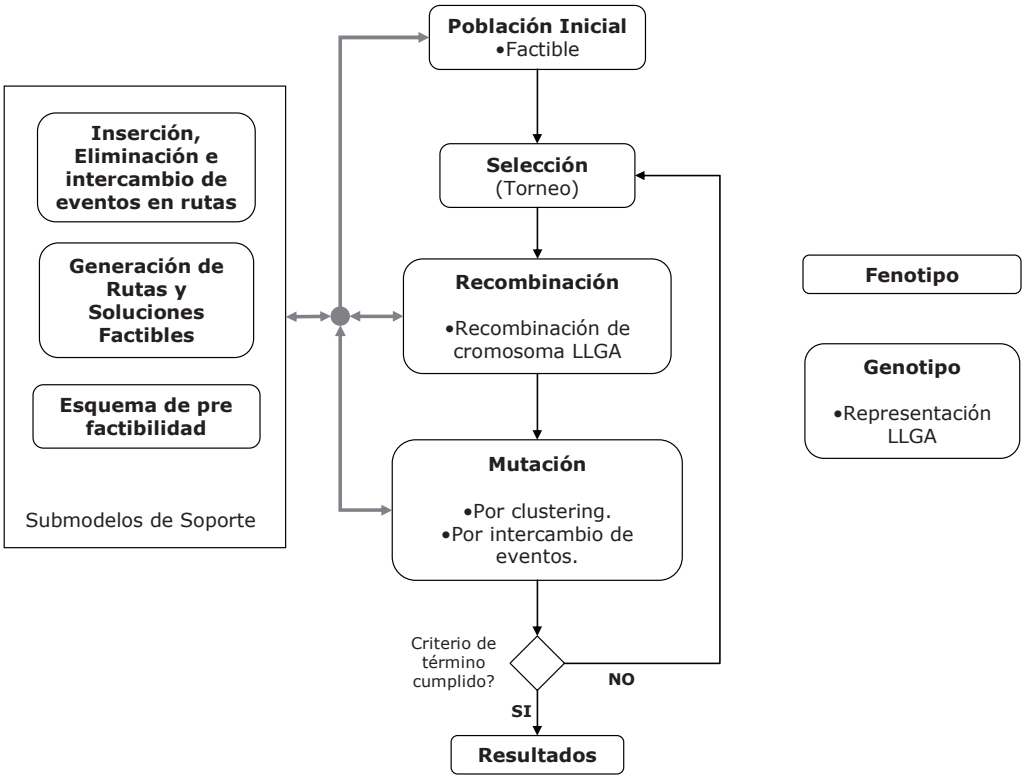


Figura 3.6 Esquema del modelo LLGA [18].

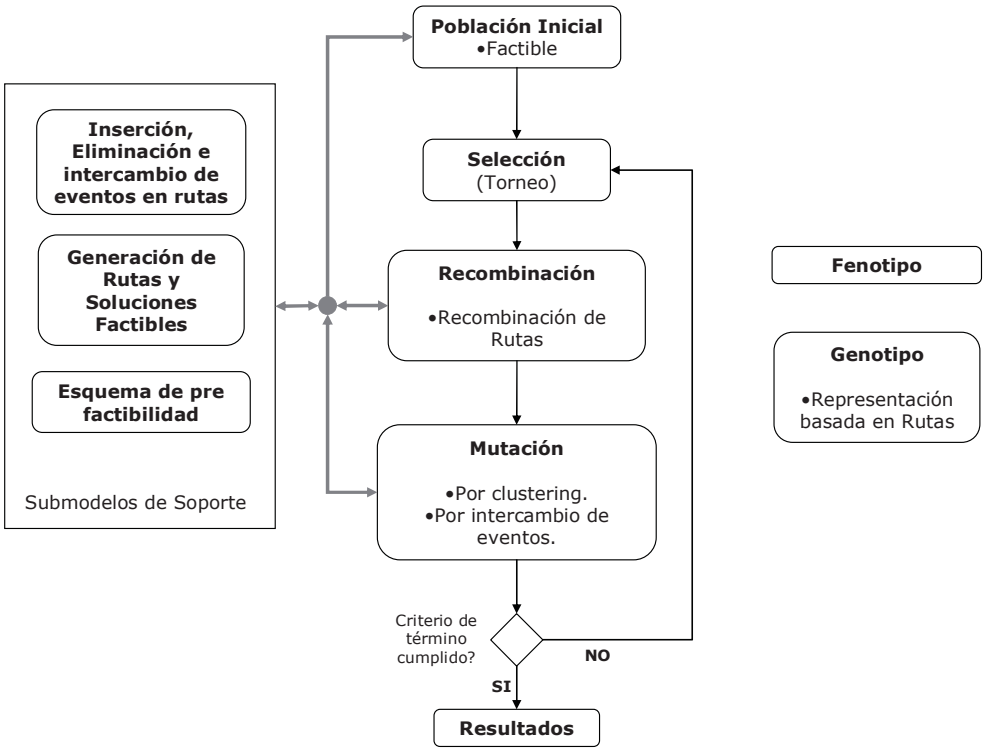


Figura 3.7 Esquema del modelo basado en rutas [18].

# 4 Sistemas Multiagente

## 4.1 Definición de Agente

No es fácil realizar una definición de agente, ya que esta palabra tiene distintos significados dependiendo del campo en el que se emplee.

Una definición de agente está dada por Russell [10] “toda entidad que de forma autónoma perciba su entorno (real o simulado) mediante sensores y que actué en el mismo mediante efectores”, siendo una de las más citadas la de Wooldrigde y Jennings [11] “un sistema informático, situado en un entorno, que es capaz de realizar acciones flexibles y autónomas para alcanzar sus objetivos de diseño”.

A pesar de lo anterior, los agentes constan de ciertas propiedades que son nombradas a continuación:

- Autonomía: los agentes son capaces de tomar decisiones sobre qué hacer sin la ayuda de un ser humano u otros agentes.
- Persistencia: el agente tiene la capacidad de tomar la iniciativa para realizar tareas que puedan llevarlo a cumplir sus metas u objetivos.
- Reactividad: un agente es capaz de percibir los cambios que se producen en su entorno y reaccionar adecuadamente, en un tiempo razonable.
- Habilidad social: los agentes tienen la capacidad de interactuar con el entorno, ya sea una sociedad artificial de agentes, como también personas humanas.

## 4.2 Sistema Multiagente

Un Sistema Multiagente (MAS) es aquel en el cual varios agentes interactúan para conseguir cierto objetivo en común o desarrollar alguna tarea. Dichos agentes trabajan de manera coordinada y tratando de optimizar el uso de los recursos que tienen a su disposición, poseen información incompleta y capacidades limitadas.

En estos sistemas el control es distribuido, los datos están descentralizados y la computación es inherentemente asíncrona.

## 4.3 Arquitecturas de Agentes

La arquitectura de un agente determina qué mecanismos utiliza éste para reaccionar a los estímulos de su entorno. Entre ellas se encuentran las que siguen a continuación:

### 4.3.1 Deliberativas

Las arquitecturas deliberativas utilizan un modelo de representación simbólica del conocimiento y sus agentes son capaces de generar planes desde un estado inicial para poder lograr sus objetivos. Por ello, se acepta la idea de que estos agentes cuenten con un sistema de planificación que pueda determinar cuáles son los pasos a seguir en la consecución de tales objetivos. Así, cada agente deliberativo cuenta con un modelo simbólico de su entorno que le permite tomar decisiones a través de un razonamiento lógico basado en la correspondencia de patrones y la manipulación simbólica.

Un ejemplo de ésta es arquitectura BDI (Belief, Desire, Intention), presentada en [12], donde cada agente cuenta con distintos estados, como creencias que representan en conocimiento del agente, deseos que representan sus objetivos e intenciones que corresponden a son deseos que el agente se compromete a realizar.

### 4.3.2 Reactivas

Esta arquitectura nace de diversos estudios sobre modelos más efectivos de representación del conocimiento y así solucionar problemas relacionados con su representación simbólica. Se caracteriza por no poseer un modelo simbólico como elemento central de razonamiento y además, por no utilizar un razonamiento simbólico complejo.

En [13] se presenta un ejemplo de esta arquitectura llamada arquitectura de subsunción, la cual se basa en la hipótesis de que la inteligencia es una propiedad emergente en ciertos sistemas complejos, por lo que se puede generar conocimiento inteligente sin necesidad de construir un modelo simbólico, sino que a través de una jerarquía de tareas organizadas en capas de distinto grado de abstracción.

### 4.3.3 Híbridas

Las arquitecturas híbridas se generan debido a las limitaciones que poseen las arquitecturas descritas anteriormente, por ello busca combinar distintos aspectos de ambas (deliberativas y reactivas).

En éstas se genera una jerarquía de capas con información del entorno, generándose tres niveles: Reactivo (de más bajo nivel) que reacciona a los estímulos del entorno y se suele implementar con una arquitectura de subsunción; Conocimiento (nivel intermedio), que posee la información del medio, por lo general mediante una representación simbólica; y Social (alto nivel), donde se manejan los aspectos sociales, ya sea con el entorno como con otros agentes.

## 4.4 Arquitecturas Multiagente

Las arquitecturas multiagente (MA) nacen de la necesidad de desarrollar aplicaciones complejas compuestas por un sin número de subsistemas que por supuesto interaccionan entre sí para lograr distribuir la inteligencia entre diversos agentes, dando origen a la creación de sistemas multiagente. El empleo de una arquitectura MA aparece como una solución apropiada cuando estamos en presencia de problemas físicamente distribuidos, o donde se requiera de experiencia heterogénea para su solución o cuando el problema en cuestión se define sobre una red de computadores. El uso de una arquitectura MA aparece en estos casos como la más adecuada en crear una solución distribuida, adaptable a cambios estructurales y de entorno. Además, una metodología asociada permitirá construir un sistema total a partir de distintas unidades autónomas.

## 4.5 Infraestructura de Comunicación de Agentes

Esta infraestructura permite definir las reglas que tendrán que cumplir los agentes para comunicarse y socializar, haciendo posible la interacción entre ellos. Consta de:

- Ontologías, las que brindan un significado común de los conceptos para conseguir el acuerdo y entendimiento entre agentes.
- Lenguajes de comunicación de agentes (ACL's), que permiten la comunicación a través de un intercambio de mensajes, contando con protocolos de comunicación definidos.
- Protocolos de interacción de agentes (AIP's), que permiten la comunicación entre agentes a través de mensajes estructurados.

### 4.5.1 Ontologías

Una ontología corresponde a una comprensión compartida de cierto dominio de interés. En ella se definen conceptos de un dominio en particular a través de un lenguaje de formulación de ontología, los que deben permitir que estas se puedan compartir, además de considerar el ciclo de evolución de éstas, apoyar la interoperabilidad, detectar inconsistencias y brindar expresividad.

La ontología toma gran importancia para conservar la interoperabilidad de sistemas que han aumentado considerablemente su tamaño.

Por lo general, una ontología posee lo siguiente:

- Clases: elementos pertenecientes a los dominios interesados.
- Propiedades: cualidades de las clases.
- Relaciones: interrelaciones entre las clases involucradas.

Los lenguajes más comunes de una ontología se señalan a continuación y se pueden ver en la Figura 4.1:

- XOL: Ontology Exchange Language, US bioinformatics community.
- SHOE: Simple HTML, Ontology Extension, University of Maryland.
- OML: Ontology Markup Language, University of Washington.
- DAML: DARPA Agent Markup Language, DARPA Project.
- OIL: Ontology Interchange Language, OntoKnowledge project.
- DAML+OIL: W3C.
- OWL: Web Ontology Language, W3C.

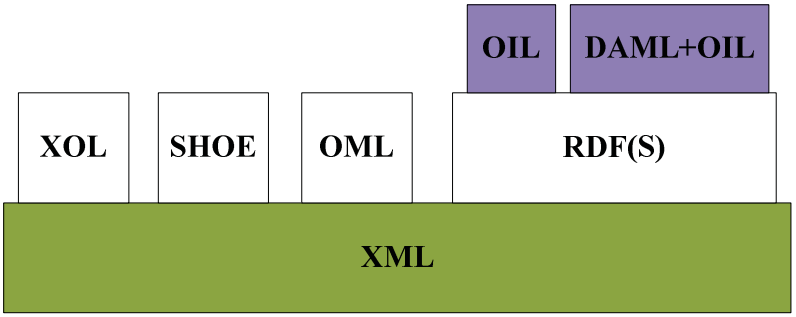


Figura 4.1 Lenguajes para ontologías [Elaboración propia].

### 4.5.2 Lenguajes de Comunicación de Agentes

Una necesidad primordial de un agente es ser capaz de entender y ser entendidos por otros, de poder reaccionar, de manera activa o pasiva, frente a alguna comunicación que se pretenda con otro agente.

Para todo lo anterior se hace necesario que los agentes compartan los siguientes aspectos:

- Sintaxis, que define la estructura de los símbolos utilizados en la comunicación y cómo se estructuran.
- Semántica, que ayuda a entender el significado de aquellos símbolos.
- Pragmática, que indica cómo deben ser interpretados los símbolos.

De esta manera, el significado incluirá tanto la semántica como la pragmática, y los agentes se podrán comunicar si se comparte una misma ontología y además, se utilice un lenguaje de comunicación que les permita intercambiar información y conocimiento.

Se ha utilizado un modelo para tratar la comunicación entre agentes, basado en la comunicación humana. En ella se distinguen los siguientes aspectos:

- Locución (sonido físico de la persona que habla).
- Ilocución (significado de lo que dijo el emisor).
- Perlocución (acto resultante de la locución).

A continuación se describen los dos principales estándares para los lenguajes de comunicación de agentes.

#### 4.5.2.1 KQML

KQML (Knowledge Query Manipulation Language) [43] es un lenguaje de comunicación y protocolo orientado a mensajes para el intercambio de información, teniendo como ventaja poseer mecanismos que permiten una traducción semiautomática desde y hacia los principales lenguajes de representación de conocimiento.

Algunos de sus elementos de mensaje se mencionan a continuación:

- Performative: tipo de mensaje.
- Sender: participante que envía el mensaje.
- Reciver: participante que debe recibir el mensaje.
- Content: contenido del mensaje.
- Language: nombre del lenguaje en el que está escrito el mensaje.
- Ontology: ontología utilizada.
- Reply-with: etiqueta para la respuesta.
- In-reply-to: etiqueta esperada en la respuesta.

#### 4.5.2.2 FIPA ACL

Este lenguaje de comunicación define la estructura que deben tener los mensajes que se envíen a los distintos agentes, presentando de manera clara los elementos prácticos de comunicación y cooperación entre ellos.

Sus elementos son similares a los de KQLM, algunos de ellos son:

- Performative
- Sender
- Receiver
- Reply-to
- Content
- Language
- Encoding
- Ontology
- Protocol
- Conversation-id
- Reply-with
- In-reply-to

- Reply-by

### 4.5.3 Protocolos de Interacción de Agentes

Los protocolos de comunicación o interacción corresponden a patrones típicos de secuencias de mensajes, en una estructura de comunicación de agentes.

FIPA ha definido ciertos protocolos a través de una notación llamada AUML, que se basa en diagramas UML, y que dan origen a los diagramas de protocolo. Además de lo anterior, existe la posibilidad de que el desarrollador pueda inscribir sus propios protocolos.

Dentro de los diagramas que posee AUML se pueden definir los siguientes elementos:

- Roles de agentes: identifican el papel de cada agente dentro de la comunicación.
- Línea de vida: define el tiempo de participación de un agente en la comunicación.
- Hitos de interacción: muestra el período de tiempo en el que un agente realiza una tarea como reacción a un mensaje entrante.
- Mensajes: acción de comunicación entre un agente y otro.

Algunos protocolos son:

- Request: a un agente se le pide que realice cierta acción.
- Request when: a un agente se le pide que realice cierta acción siempre que cumpla una precondición.
- Query: a un agente se le pide que informe sobre algo.
- Contract Net: un agente coordinador pide una propuesta a un conjunto de agentes para desarrollar una tarea, cada uno de ellos le entrega tal propuesta y el coordinador procede a escoger al agente que realizará dicha tarea.
- Brokering: un agente (broker) ofrece las funcionalidades de otros agentes o reenvía las peticiones al agente apropiado.
- Subasta Inglesa: varios agentes participan en una subasta que es iniciada con un precio bajo y que va aumentando progresivamente. A diferencia de la subasta inglesa real, puede que el precio no suba en cada ronda.
- Subasta Holandesa: varios agentes participan en una subasta que se inicia con un precio alto, que irá bajando progresivamente no necesariamente en cada ronda (a diferencia de la subasta holandesa real).
- Recruiting: funciona de la misma manera que el brokering, pero las respuestas van directamente al agente que lo necesita.
- Propose: el agente iniciador propone una serie de agentes la realización de una tarea, a la que estos pueden o no aceptar.
- Subscribe: un agente pide ser notificado si cierta condición se vuelve verdadera.

## 4.6 Metodologías de Desarrollo

Existen diversas metodologías para desarrollar sistemas multiagentes, debido a que estos difieren en gran medida de los sistemas desarrollados en otras áreas, por incluir el concepto de agente, que envuelve nociones como autonomía e inteligencia.

A continuación se describen algunas de las metodologías utilizadas para desarrollar MAS.

### 4.6.1 MESSAGE e INGENIAS

MESSAGE (Metodology for Engineering Systems of Software Agents) [41] es una metodología que incorpora técnicas de ingeniería de software, cubriendo el análisis y diseño de sistemas multiagente. Provee un lenguaje, un método y guías de cómo aplicar la metodología.

Por otro lado, la metodología INGENIAS [41] considera cinco puntos de vista mara modelar un MAS: Agente, Organización, Entorno, Tareas y Objetivos, e Interacciones.

### 4.6.2 MaSE

MaSE (Multi-agent Systems Software Engineering) [41] involucra todo el ciclo de desarrollo del sistema, desde la descripción del problema hasta su implementación. Parte del paradigma orientado a objetos, utilizando UML como notación, y asume que un agente es sólo una especialización de un objeto, la cual consiste en que los agentes se coordinan unos con otros a través de conversaciones y actúan proactivamente para alcanzar metas individuales y del sistema.

A continuación se describen las principales características de las etapas de análisis y diseño del sistema multiagente de acuerdo a esta metodología [14].

- Análisis
    - Etapas:
        - Captura de objetivos.
        - Captura de casos de uso.
        - Refinación de roles.

    - Productos obtenidos:
        - Diagramas de objetivos (requisitos funcionales del sistema).
        - Diagramas de roles (identificación de roles, tareas asociadas a ellos y comunicaciones entre roles y tareas).
        - Casos de uso.

- Diseño

- Etapas:
  - Creación de clases de agentes.
  - Construcción de conversaciones.
  - Ensamblaje se clases de agentes.
  - Diseño del sistema.

- Productos obtenidos:
  - Diagrama de clases de agente.
  - Descomposición del sistema agente en subsistemas e interconexión entre ellos.
  - Diagramas UML de despliegue que indican cuántos agentes existirían en el sistema y de qué tipo.

### 4.6.3 GAIA

La metodología GAIA [41] propone trabajar con un análisis inicial de alto nivel, en el cual se utilizan dos modelos, uno de roles y uno de interacciones. El primero identifica los roles clave en el sistema y sus propiedades, y el segundo define las interacciones a través de una referencia a un modelo de intercambio de mensajes.

Luego, se desarrollaría un diseño a alto nivel, cuyo objetivo es generar tres modelos: modelo de agentes, modelo de servicios y modelo de conocidos. El primer modelo define el tipo de agentes existentes, el número de instancias de cada tipo y los roles que cumple cada agente, el modelo de servicios identifica las funciones del agente y el último (de conocidos) define los enlaces de comunicación entre los agentes.

Principalmente, lo que busca esta metodología es especificar cómo una sociedad de agentes colabora y qué se requiere de cada uno de ellos, para satisfacer los objetivos del sistema.

### 4.6.4 PASSI

La metodología que ahora se presenta, es la que se utilizará en el desarrollo del presente proyecto, dado que es la más usada hoy en la comunidad científica de agentes.

PASSI (a Process for Agent Societies Specification and Implementation) [15] es una metodología paso a paso, que va desde los requerimientos hasta el código, para el diseño y desarrollo de sociedades multiagentes integrando modelos de diseño y conceptos desde una ingeniería de software orientada a objetos y enfoques de inteligencia artificial, utilizando notación UML ya que es ampliamente aceptado, tanto en los ambientes académicos como laborales.

Como se observa en la Figura 4.2, esta metodología se está conformada por cinco modelos referentes a diferentes niveles de diseño, y doce pasos en el proceso de construcción del sistema multiagente, como se describe a continuación:

- Modelo de Requerimientos del Sistema: Corresponde a un modelo antropomórfico de los requerimientos del sistema en términos de agencia y objetivo. Este modelo se compone de las siguientes fases:
  - Descripción del Dominio (D.D). Una descripción funcional del sistema usando diagramas de casos de uso convencionales, obteniéndose una descripción de requerimientos como la obtenida con métodos orientados a objeto.
  - Identificación de Agentes (A.Id). Se inicia a partir de los diagramas de casos de uso obtenidos en el paso previo. Permite la separación de las responsabilidades que conciernen a los agentes, representados como paquetes de UML estereotipados.
  - Identificación de Roles (R.Id). Produce un conjunto de diagramas de secuencia que especifican los escenarios más importantes del diagrama obtenido en el paso anterior. Estos diagramas son utilizados para identificar los roles interpretados por cada agente para conseguir sus objetivos.
  - Especificación de Tareas (T.Sp). En este paso se analiza un agente a la vez, representando cada uno mediante un diagrama de actividad. En este diagrama nodo de actividad representa una tarea que el agente debe realizar. De manera que, este diagrama resume el comportamiento de cada agente, así como, sus interacciones con otros. Este paso completa el modelo de requerimientos del sistema.

- Modelo de Sociedad de Agentes: Es un modelo de las interacciones y dependencias sociales entre los agentes que participan en la solución. Compuesto por cuatro fases:
  - Descripción de la Ontología de Dominio (D.O.D). Se describe la ontología del dominio mediante diagramas de clases y restricciones OCL. Este modelo describe el conocimiento atribuido a los agentes individuales y la pragmática de sus interacciones. Para describir mejor los detalles de estos elementos, se utilizan dos diagramas diferentes: la Descripción de la Ontología del Dominio y la Descripción de la Ontología de Comunicación para describir la ontología y su uso como parte de las comunicaciones y conocimiento de los agentes.
  - Descripción de Roles (R.D). Mediante un diagrama de clases se muestran los distintos roles jugados por los agentes, las tareas involucradas, las capacidades de comunicación y las dependencias inter-agente. El objetivo de este paso es modelar el ciclo de vida de cada agente, viendo los roles que juegan, la colaboración que necesitan y las comunicaciones en las que participan.
  - Descripción del Protocolo (P.D). Mediante diagramas de secuencia se especifica cada protocolo de comunicación en términos de preformativas. Para la representación de los AIP (Agent Interaction Protocol) se recomienda utilizar AUML.

- Modelo de Implementación de Agentes: Se trata de un modelo de la solución arquitectónica en términos de clase y métodos. Compuesto por dos fases:
  - Definición de la Estructura de Agentes (A.S.D). Este paso se compone de varios diagramas de clase. Los diagramas siguen dos niveles de detalle en el diseño: un nivel multi-agente y un nivel agente-único. El primero se enfoca en la arquitectura

general del sistema, describiendo agentes y sus tareas. En el segundo nivel se ve la estructura interna de cada agente, mostrando todos los atributos y métodos de la clase agente y de sus clases de tareas internas.

- Descripción del Comportamiento de los Agentes (A.B.D). Mediante diagramas de actividad/estado se describe el comportamiento individual de los agentes. Como en el paso anterior existen dos niveles de abstracción. En los diagramas multi-agente se representan flujos de eventos, los métodos invocados y los mensajes intercambiados entre los agentes. En los diagramas de agente-único se detalla la implementación de los métodos previamente usados.

- Modelo de Código: Este es un modelo de la solución a nivel de código. Compuesto por dos fases:
  - Reutilización de Código (C.R). En este paso, se intenta reutilizar patrones existentes de agentes y tareas. Aunque, los patrones no son sólo piezas de código, sino que también se incluyen piezas de diseño, pues el proceso de reutilización puede comenzar durante el diseño del sistema.
  - Completitud de Código (C.C). Se completa el código de la aplicación a partir del esqueleto producido por los patrones reutilizados.

- Modelo de Despliegue: Es un modelo de la distribución de las partes del sistema a través de las unidades de proceso del hardware y de su migración. Involucra un solo paso:
  - Configuración de Despliegue (D.C). Mediante el uso de diagramas de despliegue se describe la localización de los agentes a las unidades de procesamiento disponibles y restricciones acerca de la migración y la movilidad, donde para soportar la movilidad de los agentes se incorporan algunas extensiones de UML.

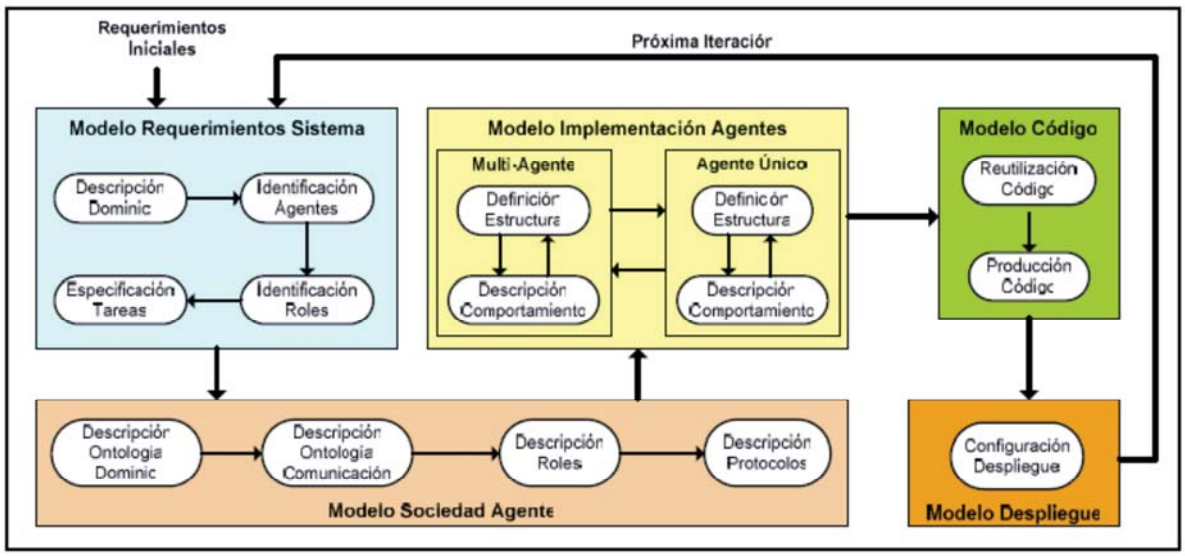


Figura 4.2 Metodología PASSI.

## 4.7 Sistemas Multiagente y Paralelización

El paralelismo se ha convertido en una herramienta computacional de gran relevancia, ya que permite dividir un problema en partes que pueden resolverse cada una en un procesador diferente, simultáneamente con las demás. Así mismo, los sistemas multiagente proveen un modelo para resolver problemas que no pueden ser resueltos de manera satisfactoria con técnicas clásicas, ya sea por el tamaño o la naturaleza del problema.

A pesar de esta particular similitud, existen también ciertas diferencias. Los MAS se han concentrado en proveer un modelo abstracto, con el cual el programador se involucra más con el modelado de un problema y la lógica del programa, que con los detalles de la ejecución paralela en sí misma. Por su parte, los desarrollos de paralelismo se han orientado más hacia la eficiencia, siendo la mayoría programas de cálculo numérico.

### 4.7.1 Paralelización con un modelo de Sistema Multiagente

En un diseño de sistemas paralelos se tiene en cuenta lo siguiente [16]:

- Descomposición (dividir el problema y la solución en partes más pequeñas).
- Comunicación (cómo se lleva a cabo la comunicación entre los distintos procesos o computadoras)
- Sincronización (orden de ejecución de los componentes)

Sin embargo, existen ciertas dificultades que generalmente se encuentran en una programación en paralelo: dividir, de manera eficiente y efectiva, el trabajo entre los distintos componentes de software; diseñar e implementar una comunicación apropiada entre los componentes; implementar la coordinación en la ejecución; y el manejo de excepciones, errores y posibles fallas.

#### 4.7.1.1 Solución a partir de un Sistema Multiagente

Un sistema multiagente puede resolver las dificultades mencionadas en la sección 3.7.1, de manera natural, ya que aquellos aspectos son manejados implícitamente por las capacidades de los agentes. La descomposición ocurre en el momento de asignar al agente una o más tareas, y los problemas de sincronización se ven reducidos debido a que la racionalidad propia del agente le indica cuándo debe y/o puede realizar una acción.

Por lo anterior, se puede percibir que la utilización de un sistema multiagente en un programa en paralelo es más fácil de mantener y mejorar, y corresponde a un importante cambio de paradigma, dado que simplifica la tarea del programador, a la hora de pensar la solución del problema.

# 5 Trabajos relacionados

## 5.1 DARPTW

### 5.1.1 Cubillos C., Urra E., Rodríguez N.

En la investigación realizada en [18], que es la base del presente trabajo, se propone el estudio de algoritmos genéticos para el DARPTW.

Se desarrollaron dos de modelos de GA, uno está basado en la adaptación de un modelo genérico, llamado Linkage Learning Genetic Algorithm y el otro corresponde a una propuesta basada en operadores personalizados. En ambos modelos, se aplicaron técnicas de pre-procesamiento en los datos de entrada, con el fin de obtener información útil en lo que se refiere al manejo de restricciones del problema, y facilitando con ello la búsqueda de soluciones y utilización de operadores en un contexto factible. Además se incluyen las operaciones propias del algoritmo genético, que en este caso son:

- Genotipo, LLGA o basado en rutas, dependiendo del modelo.
- Fenotipo, que se realiza a través de la función objetivo del problema.
- Población inicial, para la cual se utiliza la herramienta de generación de soluciones factibles que se explica en la sección 3.12.3.
- Selección, por torneo (ver sección 2.2.4).
- Recombinación, de cromosoma LLGA o de rutas, dependiendo del modelo.
- Mutación, por clustering o intercambio de eventos.

Como datos de entrada se consideraron dos conjuntos que se han utilizado en otros estudios, uno para un algoritmo de Branch-and-Cut y el otro para un algoritmo de búsqueda Tabú. Las comparaciones se realizaron entre los dos modelos propuestos y además se utilizó el estudio de Bergvinsdottir, Larsen and Jorgensen [21] (Algoritmo genético cluster-first route-second) y el de Cordeau y Laporte (Tabu Search) [42].

Los resultados obtenidos mostraron que el modelo basado en LLGA entrega mejoras interesantes en lo que se refiere al tiempo de ejecución del algoritmo, además que los dos modelos desarrollados mejoraron en cierto nivel las prestancias de la calidad de servicio en las instancias testeadas.

## 5.2 Algoritmos Genéticos Paralelos

### 5.2.1 Zhao T., Man Z., Qi X.

En [37] se propone un algoritmo genético paralelo híbrido grano grueso maestro-esclavo (CGS-MSM PGA) basado en un sistema multiagente, cuya estructura es la misma que la de dicho PGA (ver sección 2.3.1.4). Los maestros son representados por agentes llamados *managed Agent* (M-Agent) y los esclavos por los agentes *algorithm Agent* (AAgent), como se puede ver en la Figura 5.1.

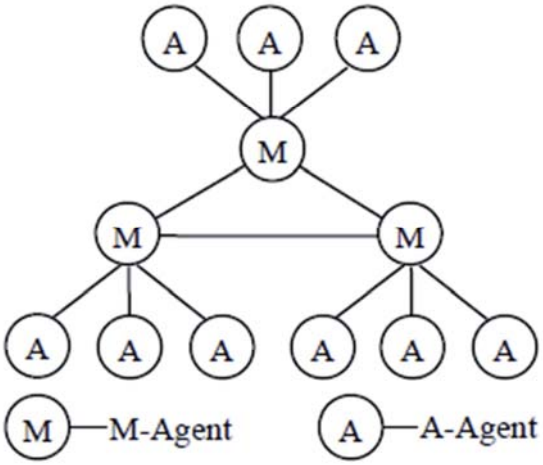


Figura 5.1 Estructura de CGS-MSM PGA basado en un sistema multiagente.

La resolución del problema sigue una serie de pasos que son mencionados a continuación:

- Paso 1: Cada agente M-Agent se inicializa; se genera la población inicial; se termina la comprobación de la restricción de factibilidad; se calcula el fitness; se realiza la selección, generación de una nueva población, y se envía a cada agente A-Agent.
- Paso 2: Cada A-Agent calcula el fitness para casa individuo en la nueva población, realiza la operación de cruce y genera otra población; cada A-Agent envía la nueva población a su agente M-Agent correspondiente; se recibe la nueva población enviada por A-Agent y se selecciona la mejor.
- Paso 3: Todos los M-Agent intercambian entre sí el mejor individuo y de genera una nueva población.
- Paso 4: Cada M-Agent toma la nueva población como la población inicial para realizar nuevamente los pasos mencionados.

Se realizó una comparación utilizando los mismos datos de otros estudios, que habían dado como mejor resultado hasta el momento una distancia más corta de 15768 KM. En este

caso el largo total de la ruta fue de 15745 KM, generando un mejor resultado que las investigaciones anteriores.

# 6 Modelos propuestos

En este capítulo se presenta el modelo general del sistema a desarrollar. Este se basa en el trabajo realizado en [18] donde se propone una solución al DARPTW utilizando Algoritmos Genéticos. A continuación, en la sección 6.1, se muestra el detalle del marco de trabajo propuesto en esta investigación.

## 6.1 Marco de trabajo

Se desea implementar un MAS para paralelizar la solución basada en Algoritmos Genéticos para el DARPTW. Para lo anterior se propone utilizar un algoritmo genético paralelo Maestro-Esclavo modificado (se explicará en la sección 6.1.3), el cual difiere del Maestro-Esclavo clásico (descrito en la sección 2.3.1.1) en que el maestro realizará la generación de la población inicial y la selección, y los esclavos desarrollarán las operaciones de recombinación y mutación. En el desarrollo de esta sección se irá detallando la manera en que se utilizará este modelo.

En la sección 3.12 se trató parte de lo realizado en la investigación en que se basa el trabajo actual. En éste se presentan dos modelos de algoritmos genéticos para la resolución del problema, uno basado en la adaptación de un modelo genérico llamado Linkage Learning Genetic Algorithm (LLGA) y otro correspondiente a una propuesta que utiliza operadores personalizados basados en ruta. De forma transversal a ambos modelos, se aplican técnicas de pre-procesamiento en los datos de entrada, con el fin de obtener información útil en lo que se refiere al manejo de restricciones del problema, facilitando con ello la búsqueda de soluciones y utilización de operadores en un contexto factible. De manera general, ambos modelos se inician con la obtención de una población inicial factible, utilizando la herramienta de generación de soluciones factibles (ver sección 3.12.3). Luego se realiza un proceso de selección basado en torneo (explicado en la sección 2.2.4), continuando con la operación de recombinación de cromosoma LLGA, en uno de los modelos, y recombinación de rutas en el otro. Posteriormente se realiza la operación de mutación, ambas operaciones, recombinación y mutación, se llevan a cabo de acuerdo a ciertos criterios basados en probabilidades. El proceso finaliza con la evaluación del criterio de término, el cual generalmente se refiere a un determinado número de generaciones.

### 6.1.1 Modelo LLGA versus Modelo basado en rutas

De acuerdo a los resultados obtenidos en [18], el modelo LLGA mostró un mejor rendimiento en tiempos, comparándolo con el modelo basado en rutas en todos los casos estudiados. Lo anterior se debe a que el operador de recombinación del modelo LLGA trabaja en una sola etapa, a diferencia del operador del otro modelo mencionado, en el cual se realizan tres etapas que tienen un costo de tiempo considerable reflejado en los resultados de dicha investigación.

Por lo recién mencionado, el presente trabajo se enfocará en el primer modelo, ya que lo que se persigue principalmente al paralelizar el problema es justamente disminuir tiempos de ejecución. De esta forma, se utilizarán los mismos elementos y criterios para así tener un

referente real con el cual, a través de futuras comparaciones, se concluirá si lo que se planea implementar puede, en cierta medida, mejorar los resultados antes obtenidos.

### 6.1.2 Implementación en Java

Dentro del material con que se cuenta, se encuentra el código realizado en [18], que entrega las soluciones estudiadas en la investigación anterior. Dicho código se encuentra en lenguaje de programación C#, con lo cual no se podría trabajar directamente sobre la plataforma JADE, ya que ésta es una herramienta de desarrollo de sistemas multiagente que está totalmente implementada en Java.

Como solución a lo planteado anteriormente, se decidió realizar una migración del código desde un lenguaje a otro, manteniendo la lógica utilizada, cambiando implementaciones de ciertas rutinas que funcionan de distinta manera en cada lenguaje y creando algunas clases utilizadas en el código C#, que son propias del sistema y no se encuentran en las librerías de Java. En esto último es donde se encuentran la mayoría de las limitaciones al momento de realizar la migración, por ejemplo, los tipos de datos que no existen en Java pero si en C#, la implementación de delegados, la utilización de parámetros por referencia (sentencia out en C#), entre otros. Cabe destacar que una aplicación .NET no es transportable en un 100% a Java, ya que .NET está hecho sobre y exclusivamente para Windows, por lo tanto asume y está orientado para usar recursos de esta plataforma. Java, en cambio es multiplataforma y posiblemente no existan consolas en todos los sistemas operativos, por eso es más limitado.

Debido a lo mencionado en el párrafo anterior, se espera que los resultados obtenidos en esta implementación, sean soluciones que no se desvíen mayormente de las soluciones originales. Esto se probará utilizando los mismos archivos de entrada para analizar datos iguales. De esta manera, una vez lista la implementación, se estaría en condiciones de planear la distribución del algoritmo en el sistema multiagente que se describe a continuación.

### 6.1.3 Descripción del MAS

Como se explicó al comienzo de este capítulo, lo que se persigue en el presente trabajo es paralelizar la solución del problema. Para esto se ha escogido utilizar un Sistema Multiagente que distribuya el algoritmo, utilizando el modelo de un Algoritmo Genético Paralelo Maestro Esclavo con ciertas modificaciones que se describen a continuación.

La Figura 6.1 presenta un esquema que refleja el comportamiento de los agentes (maestro y esclavos) dentro del sistema a construir. El agente maestro debe configurar los parámetros correspondientes al GA para luego obtener la población inicial factible. Con ésta debe entregar las primeras estadísticas de acuerdo a dicha generación, ya que luego realizará la selección por torneo para obtener una generación intermedia que enviará a los agentes esclavos en cantidades iguales, según el tamaño de la población y la cantidad de esclavos que existan en el sistema.

Por otro lado, el agente esclavo realizará todas las operaciones de recombinación y mutación. Para esto, recibirá las porciones de las generaciones intermedias que vaya obteniendo el agente maestro y le aplicará los dos operadores mencionados anteriormente de acuerdo a ciertas condiciones, las cuales influyen en la generación de soluciones factibles.

A nivel de implementación, para la recombinación LLGA (explicada en la sección 3.12.5) se utiliza un arreglo de cromosomas padres que se comportan, uno como donante de un segmento y el otro como receptor escogiendo en él un punto de inserción. Se realizan dos instancias de recombinación, donde se intercambian los roles del donante y receptor de acuerdo a dichos padres. Para cada una de las instancias, si se obtienen resultados infactibles, se realizan intentos de recombinación hasta obtener resultados factibles.

En el caso la mutación de clúster LLGA se evalúa las posibilidades de mutación de todos los genes en el cromosoma, revisando cada uno de ellos para comprobar que cumple con la probabilidad. Si se cumple, se sigue desarrollando el proceso normal de este tipo de mutación explicado en la sección 3.12.7, en caso contrario se continúa con el próximo gen. Por otro lado, la mutación de rutas LLGA evalúa las posibilidades de mutación para todas las rutas internas del cromosoma. Se va revisando cada ruta, si no cumple la probabilidad o no contiene puntos intercambiables se prosigue con la siguiente ruta, sino se realiza el proceso normal de este tipo de mutación explicado en la sección 3.12.7. Ninguno de los procesos mencionados anteriormente asegura que estos dos tipos de mutación ocurran, si la mutación no es factible ya sea por probabilidad u otra condición, entonces no se realiza.

Luego de aplicar las operaciones mencionadas anteriormente, cada agente esclavo enviará al agente maestro la nueva generación obtenida para que éste reúna cada una de ellas y vuelva a repetir los procesos de selección y obtención de generaciones intermedias.

Todo el proceso explicado se convierte en un ciclo en el que el agente maestro obtiene una generación intermedia que divide en porciones para ser enviadas a los agentes esclavos, estos realizan las operaciones correspondientes y devuelven la próxima generación, que es recibida por el maestro de parte de cada uno de los esclavos para así reunirlas y generar las estadísticas correspondientes a dicha generación. Luego de esto, el agente maestro vuelve a obtener una generación intermedia, repitiéndose los mismos pasos explicados y finalizándose cuando se completa el número máximo de generaciones, existiendo resultados de evaluación para cada generación obtenida.

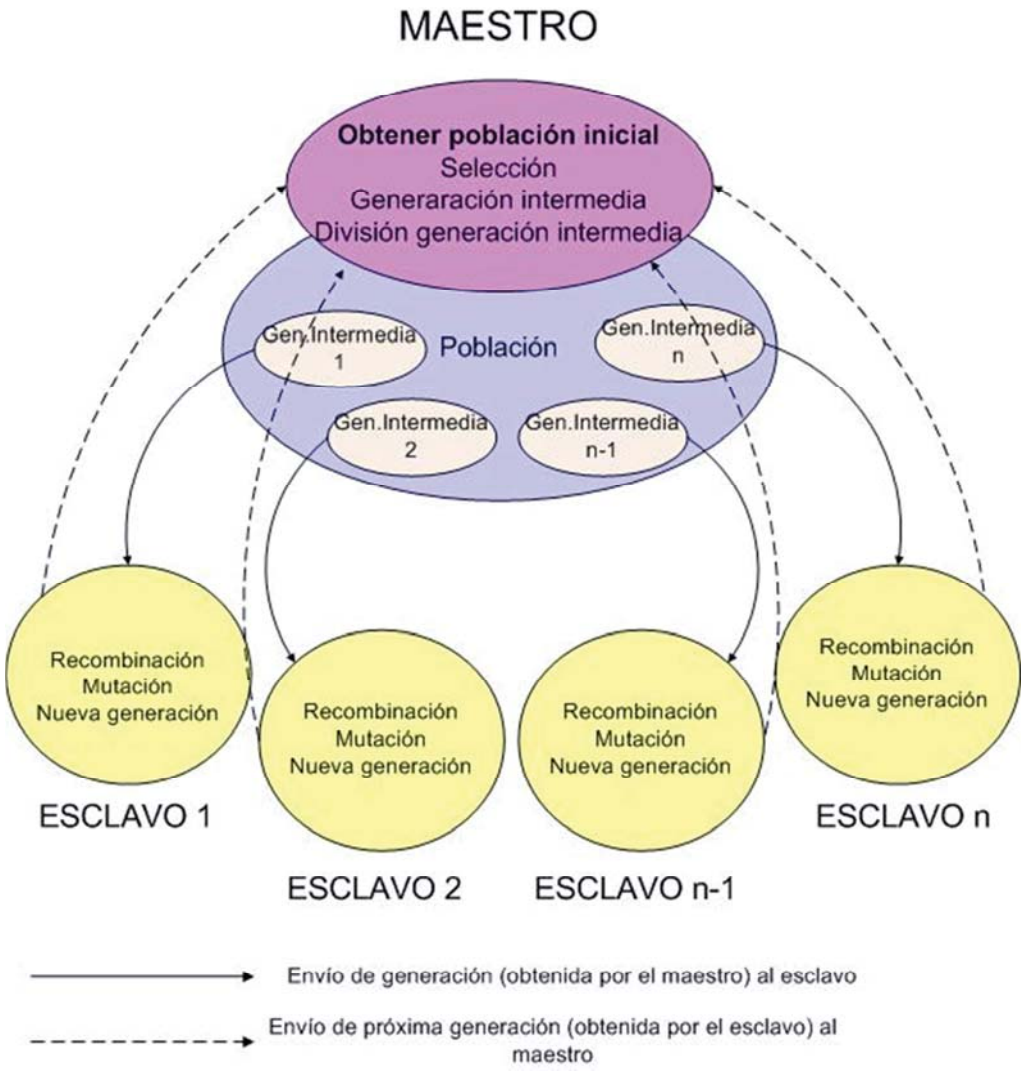


Figura 6.1 Esquema de comportamiento de los agentes en el MAS [Elaboración propia].

Ambos agentes se comunicarán a través de mensajes de información para enviar objetos de uno a otro, evitando un excesivo intercambio de mensajes ya que significaría una utilización de tiempo importante.

# 7 Diseño del sistema

En este capítulo se especifica el diseño del sistema multiagente a implementar según lo presentado en el marco de trabajo explicado en el capítulo anterior. Se consideran modelos para el esquema del modelo LLGA por los motivos explicados en la sección 6.1.1.

Para el diseño del sistema se utilizará metodología PASSI cuyas fases y pasos se muestran a continuación.

## 7.1 Modelo de Requerimientos del Sistema

### 7.1.1 Descripción del Dominio

En general, el dominio está representado por actividades propias de un algoritmo genético simple, como la realización de la selección, recombinación y mutación, la evaluación de calidad de los individuos.

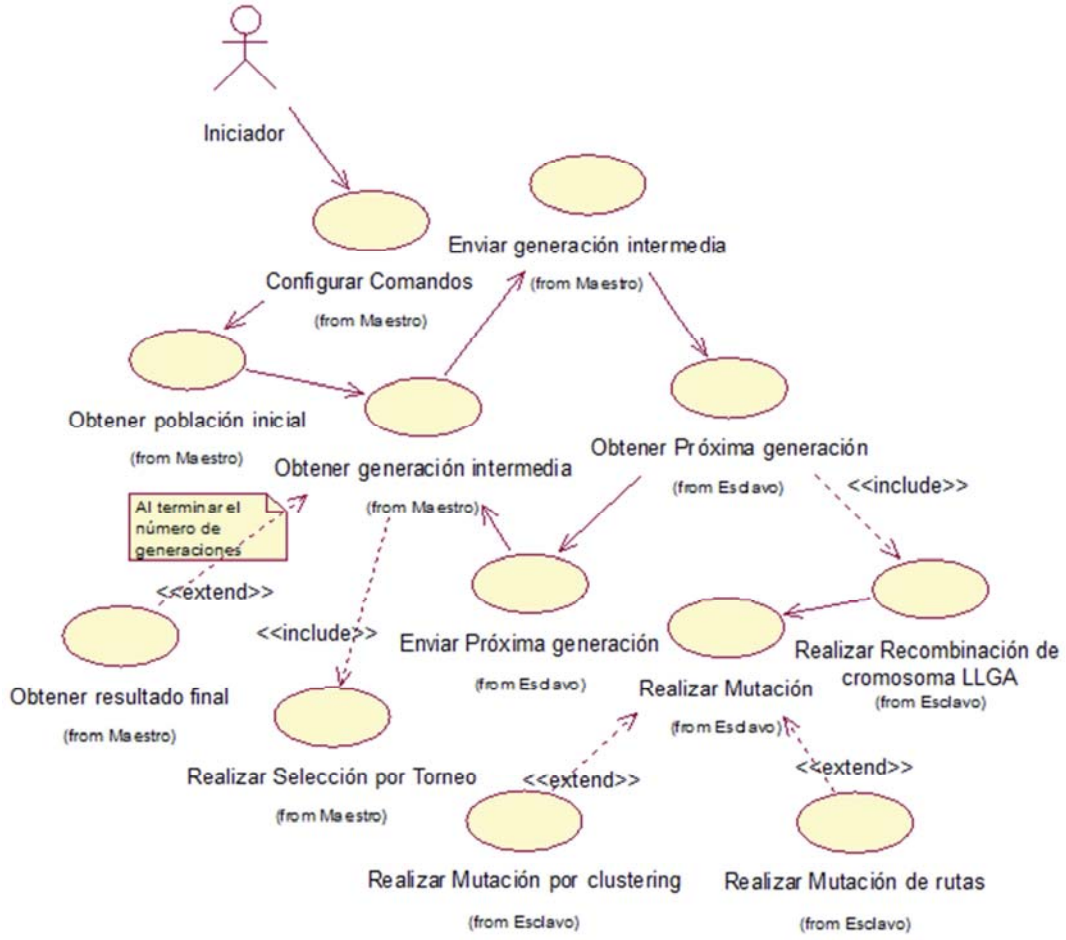


Figura 7.1 Descripción del Dominio: Modelo LLGA.

### 7.1.2 Identificación de Agentes

Se identifican 2 agentes:

- Maestro: este agente es el encargado de comenzar la ejecución principal y configurar los parámetros que se utilizarán en la ejecución del GA. Debe obtener la población inicial con que se trabajará, para luego realizar la selección y obtener las generaciones intermedias que deben ser enviadas a los agentes esclavos durante el ciclo de vida del algoritmo.
- Esclavo: el agente esclavo será el encargado de realizar las operaciones de recombinación y mutación para obtener la próxima generación a evaluar.

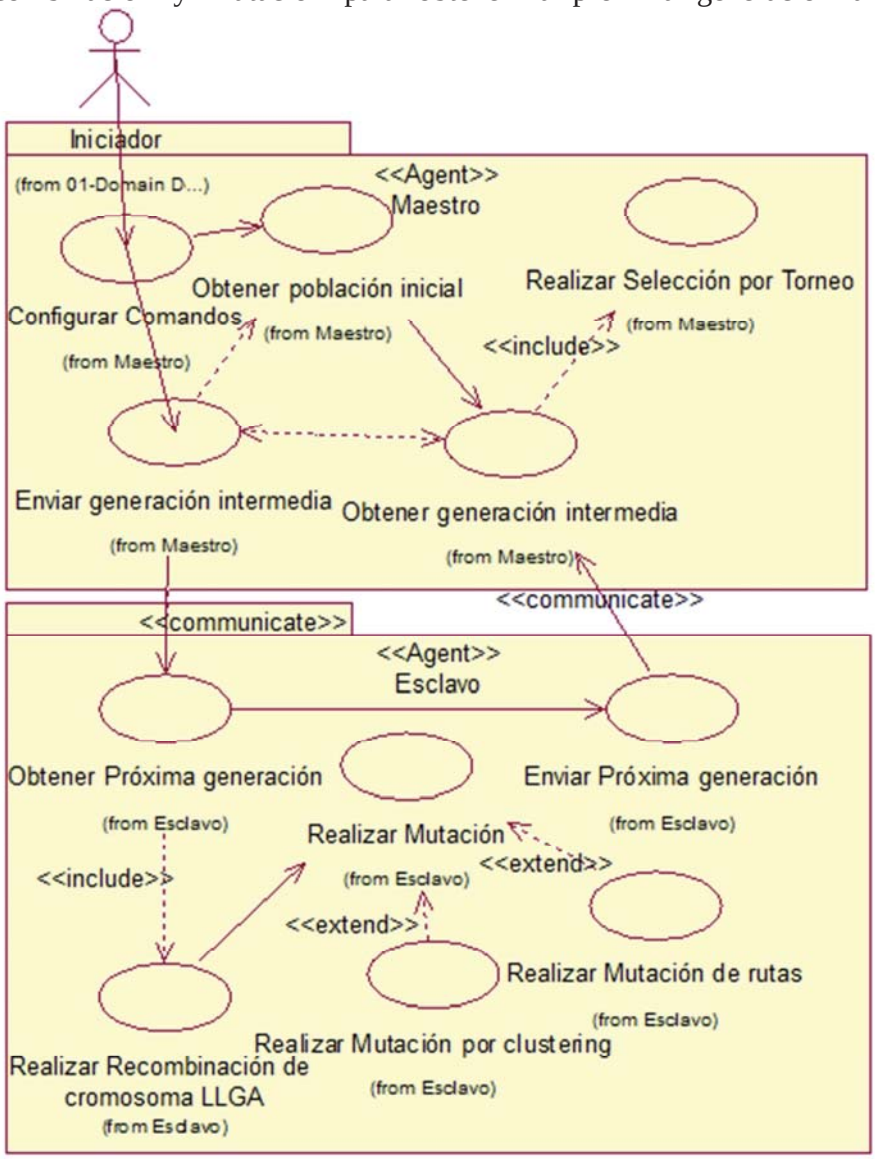


Figura 7.2 Identificación de Agentes: Modelo LLGA.

### 7.1.3 Identificación de Roles

A continuación se describen algunos de los escenarios que estarán presentes en la implementación del sistema.

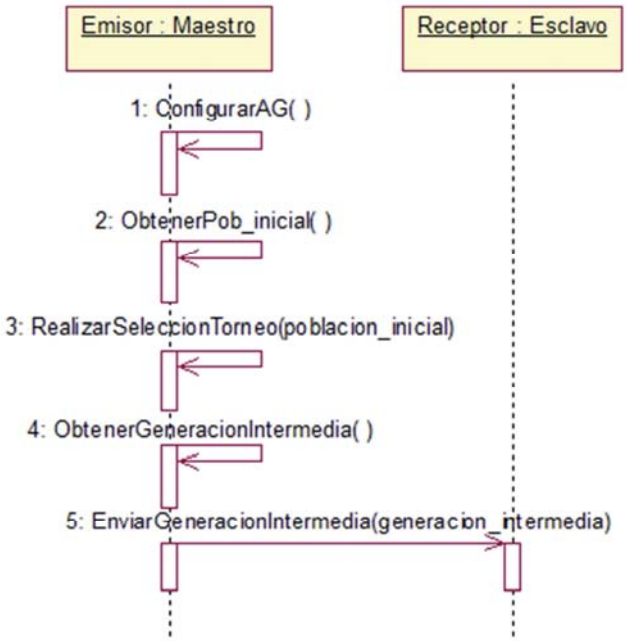


Figura 7.3 Identificación de Roles. Escenario: “El maestro envía generación intermedia a un esclavo”.

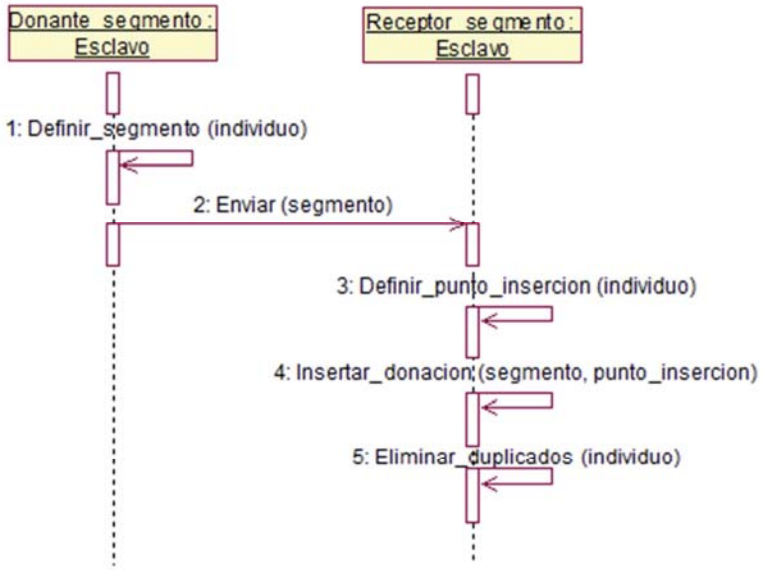


Figura 7.4 Identificación de Roles. Escenario: “El esclavo realiza la operación de recombinación según el modelo LLGA”.

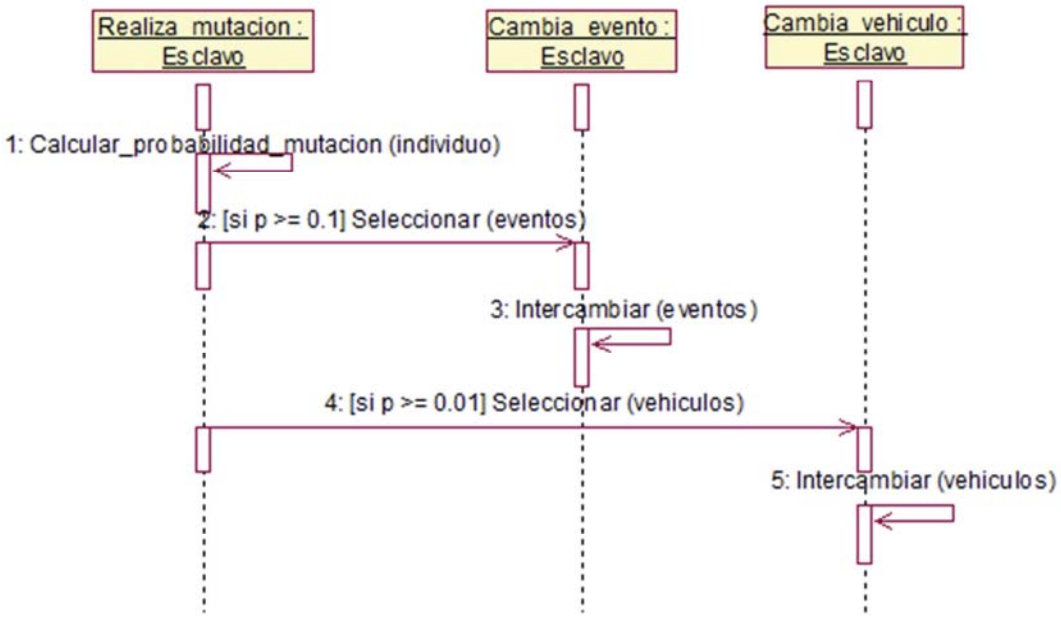


Figura 7.5 Identificación de Roles. Escenario: “El esclavo realiza la operación de mutación”.

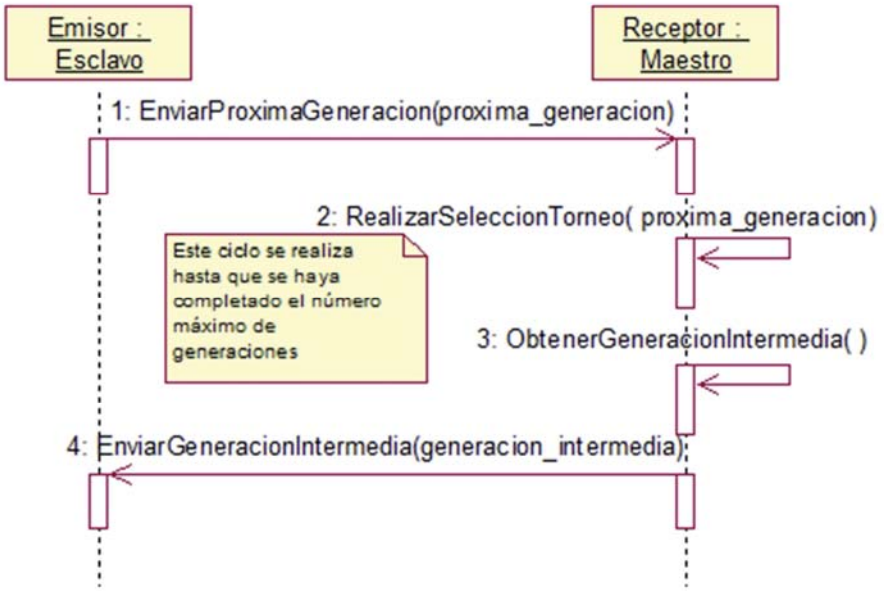


Figura 7.6 Identificación de Roles. Escenario: “Se realiza ciclo entre esclavo y maestro”.

### 7.1.4 Especificación de tareas

A continuación se presentan algunos de los diagramas de actividad que podrían reflejarse en el sistema.

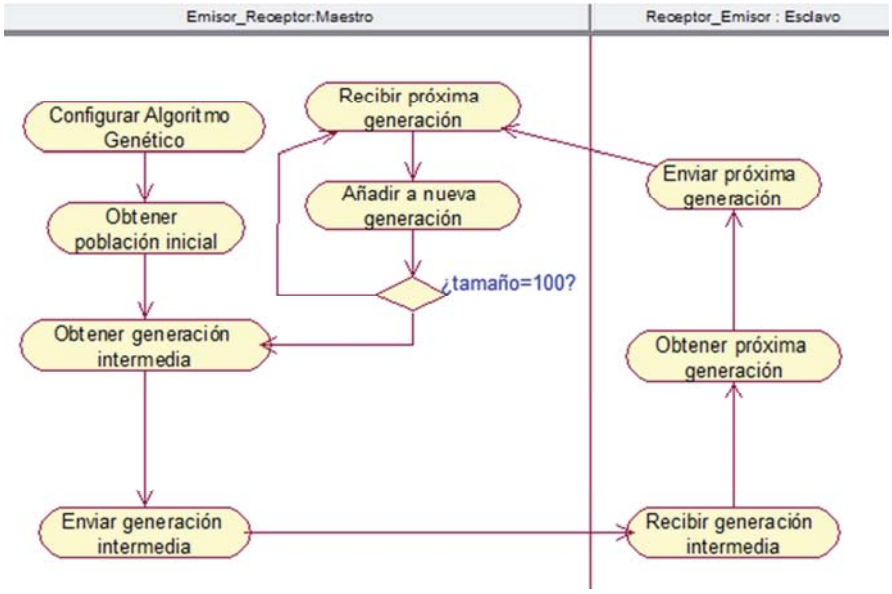


Figura 7.7 Maestro-Esclavo: Ciclo de actividades entre maestro y esclavo.

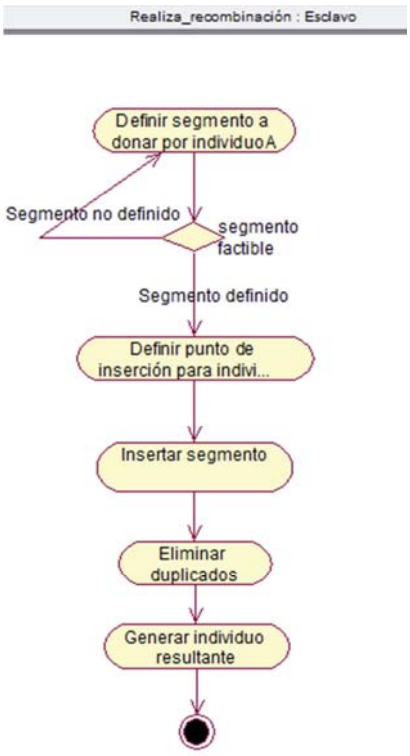


Figura 7.8 Esclavo: Recombinación según modelo LLGA

## 7.2 Modelo de Sociedad de Agentes

### 7.2.1 Descripción de la ontología de dominio

En la Figura 7.9 se presenta el diagrama de descripción de la ontología de dominio, cuyo objetivo es mostrar el conocimiento que se atribuye a cada agente y la comunicación entre ellos.

### 7.2.2 Descripción de roles

El diagrama de descripción de roles se muestra en la Figura 7.10. Este es un diagrama de clases que contiene paquetes que representan a los agentes y son los contienen los roles, los que a su vez se pueden conectar por ciertas relaciones.

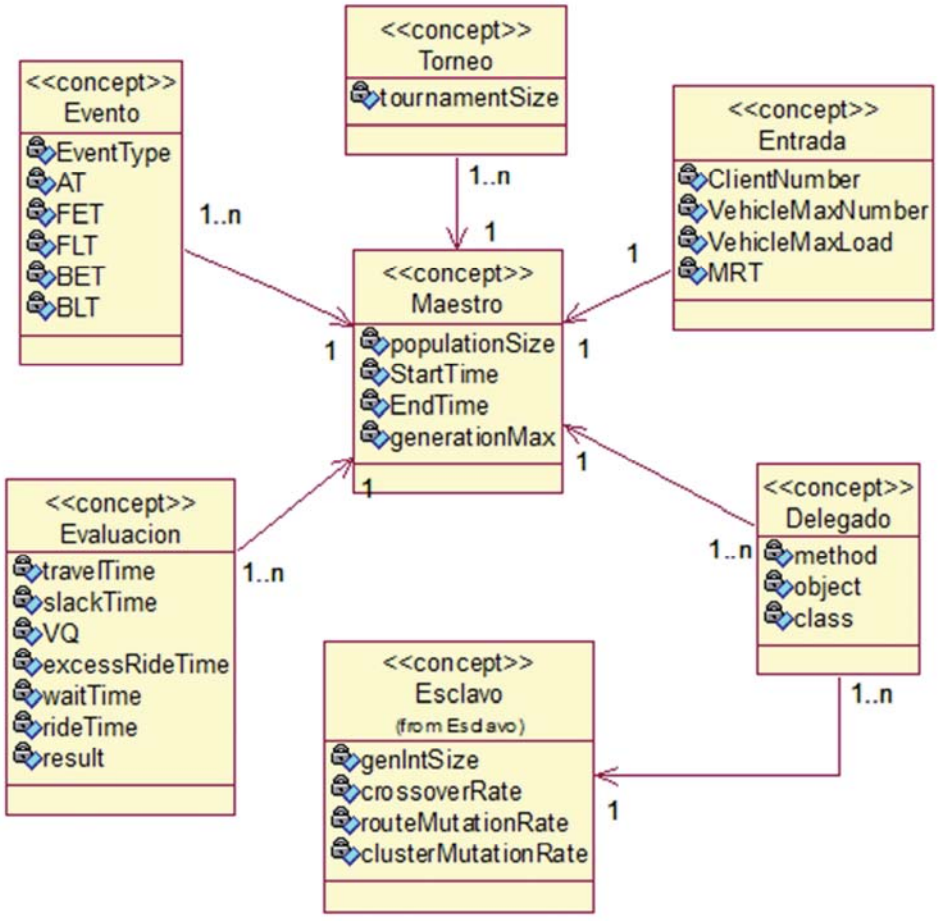


Figura 7.9 Diagrama de descripción de Ontología de Dominio

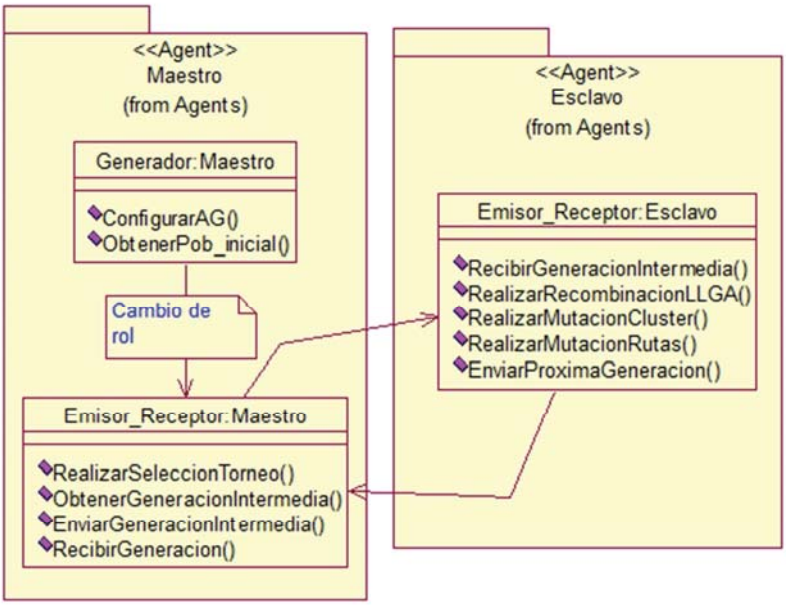


Figura 7.10 Diagrama de descripción de Roles

## 7.3 Modelo de Implementación de Agentes

### 7.3.1 Datos necesarios al interior de los agentes

- Agente Maestro

  Debe tener los datos que se cargan desde el archivo de entrada que contiene los datos del problema además de los valores relacionados con los parámetros que se utilizarán, como el tamaño de la población, número máximo de generaciones, entre otros.

- Agente Esclavo

  Debe contener los datos relacionados a las operaciones de recombinación y mutación, como por ejemplo, lo ratios de cada uno.

### 7.3.2 Descripción de la Estructura de agentes

#### 7.3.2.1 Definición de la Arquitectura Multiagente

En la Figura 7.11 se muestra el diagrama para la definición de la arquitectura multiagente, el cual contiene al actor y las clases involucradas que corresponden a los agentes que se utilizarán en el sistema. Se encuentran también los atributos y operaciones de cada agente, además de la comunicación que existe entre ellos.

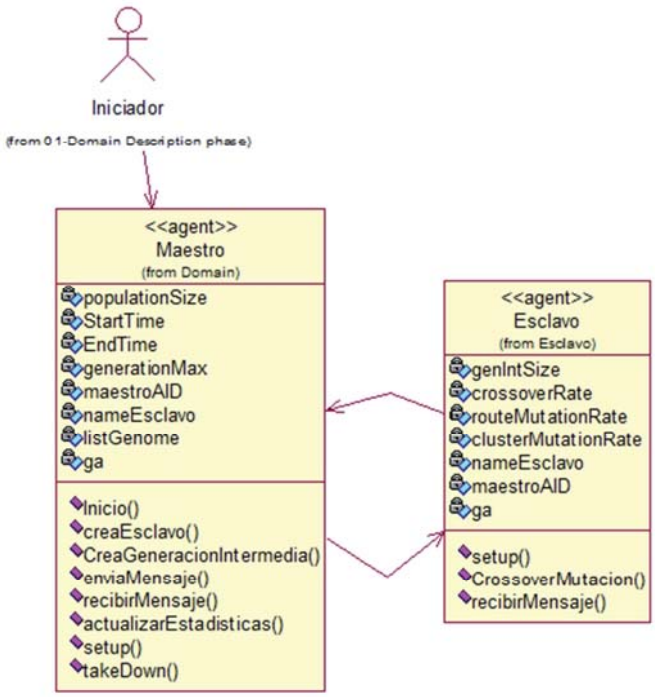


Figura 7.11 Definición de la Arquitectura Multiagente

### 7.3.2.2 Definición de la estructura del agente individual

- Agente Maestro

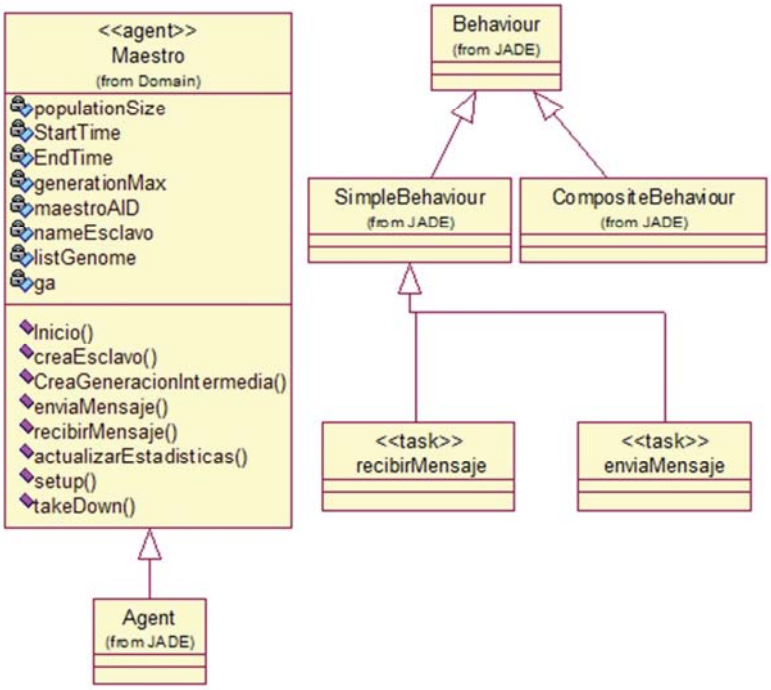


Figura 7.12 Estructura del Agente Maestro

- Agente Esclavo

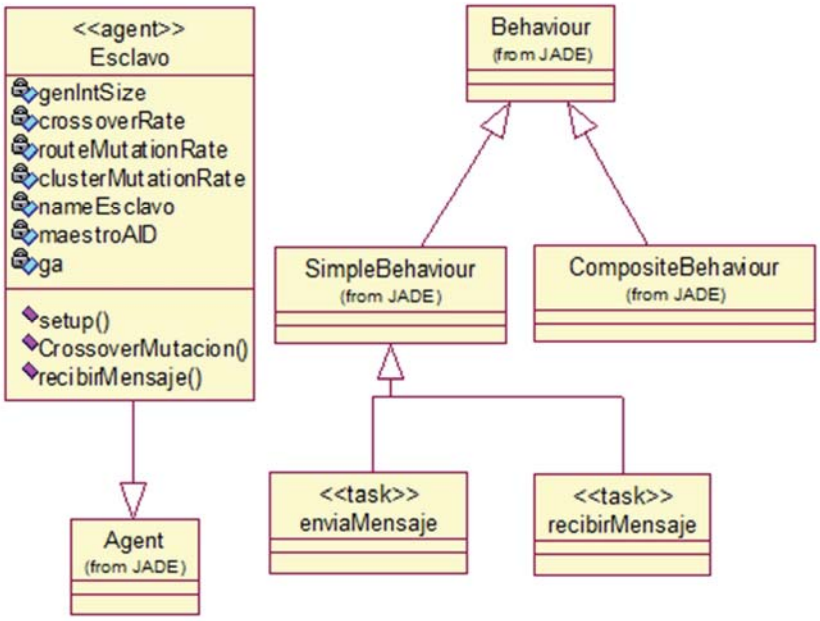


Figura 7.13 Estructura del Agente Esclavo

### 7.3.3 Archivos de entrada y salida

Se tienen cuatro archivos, dos de entrada y dos de salida.

Los archivos de entrada serán del mismo tipo que los utilizados en [18], para así poder realizar comparaciones al finalizar esta investigación. Estos se pueden visualizar en las Figuras 7.14 y 7.15, y corresponden a datos de entrada que se han utilizado en ciertos estudios. Han sido divididos en dos subconjuntos, uno utilizado para un algoritmo de Branch-and-Cut y el otro para uno de búsqueda Tabú. Para la implementación actual, los archivos se modificarán separando cada elemento por comas como se muestra en la Figura 7.16.

```
2 16 480 3 30
  0    0.000    0.000    0    0     0  480
  1   -1.198   -5.164    3    1     0 1440
  2    5.573    7.114    3    1     0 1440
  3   -6.614    0.072    3    1     0 1440
  4   -7.374   -1.107    3    1     0 1440
  5   -9.251    8.321    3    1     0 1440
  6    6.498   -6.036    3    1     0 1440
  7    0.861    6.903    3    1     0 1440
  8    3.904   -5.261    3    1     0 1440
  9    7.976   -9.000    3    1   276  291
 10   -2.610    0.039    3    1    32   47
 11    4.487    7.142    3    1   115  130
 12    8.938   -4.388    3    1    14   29
 13   -4.172   -9.096    3    1   198  213
 14    7.835   -9.269    3    1   160  175
 15    2.792   -7.944    3    1   180  195
 16    5.212    9.271    3    1   366  381
 17    6.687    6.731    3   -1   402  417
 18   -2.192   -9.210    3   -1   322  337
 19   -1.061    8.752    3   -1   179  194
 20    6.883    0.882    3   -1   138  153
 21    5.586   -1.554    3   -1    82   97
 22   -9.865    1.398    3   -1    49   64
 23   -9.800    5.697    3   -1   400  415
 24    1.271    1.018    3   -1   298  313
 25    4.404   -1.952    3   -1     0 1440
 26    0.673    6.283    3   -1     0 1440
 27    7.032    2.808    3   -1     0 1440
 28   -0.694   -7.098    3   -1     0 1440
 29    3.763   -7.269    3   -1     0 1440
 30    6.634   -7.426    3   -1     0 1440
 31   -9.450    3.792    3   -1     0 1440
 32   -8.819   -4.749    3   -1     0 1440
 33    0.000    0.000    0    0     0  480
```

Figura 7.14 Archivo de entrada Branch-and-Cut 16 clientes

```
3 48 480 6 90
  0   -1.044    2.000  0   0     0 1440
  1   -2.973    6.414 10   1     0 1440
  2   -3.066    0.546 10   1     0 1440
  3    5.164    0.547 10   1     0 1440
  4   -1.317    6.934 10   1     0 1440
  5   -6.741    6.832 10   1     0 1440
  6    4.891    0.627 10   1     0 1440
  7    0.524    2.226 10   1     0 1440
  8   -6.500    7.723 10   1     0 1440
  9   -0.417   -0.157 10   1     0 1440
 10    2.303    1.164 10   1     0 1440
 11    2.548    0.629 10   1     0 1440
 12   -4.261   -2.639 10   1     0 1440
 13   -7.667    9.934 10   1   325  358
 14   -2.067    5.789 10   1   111  152
 15   -5.204    0.657 10   1   395  421
 16   -4.138    5.082 10   1   386  401
 17   -9.194    2.759 10   1    86  114
 18   -6.512    3.021 10   1   409  426
 19    1.860    9.672 10   1   454  470
 20   -4.094    8.321 10   1   175  202
 21   -3.776   -3.333 10   1   416  453
 22    2.377    2.908 10   1   147  177
 23   -4.303    2.045 10   1   471  499
 24   -3.530   -2.490 10   1   321  346
 24   -3.530   -2.490 10   1   321  346
 25   -5.476    1.437 10  -1   258  287
 26   -4.933    3.337 10  -1   329  361
 27    5.740    2.382 10  -1   209  252
 28   -2.275    5.541 10  -1   416  460
 29   -5.662    7.334 10  -1   305  349
 30   -3.856   -0.370 10  -1   432  458
 31   -1.678    1.954 10  -1   202  236
 32   -1.156    1.161 10  -1   225  252
 33   -4.655    9.797 10  -1   102  123
 34    1.623    0.932 10  -1   260  276
 35    0.129    0.735 10  -1   178  215
 36   -2.640    2.953 10  -1   381  397
 37    0.435    1.469 10  -1     0 1440
 38   -5.066   -2.313 10  -1     0 1440
 39   -2.283   -0.981 10  -1     0 1440
 40   -7.110   -1.862 10  -1     0 1440
 41   -0.785    3.207 10  -1     0 1440
 42    1.188   -2.493 10  -1     0 1440
 43   -1.893   -2.373 10  -1     0 1440
 44   -1.192    1.175 10  -1     0 1440
 45    2.984    1.163 10  -1     0 1440
 46    1.227   -5.581 10  -1     0 1440
 47   -3.793   -2.161 10  -1     0 1440
 48    4.288   -0.297 10  -1     0 1440
```

Figura 7.15 Archivo de entrada Tabu Search 24 clientes

```
2;16;480;3;30
0;0.000;0.000;0;0;0;480
1;-1.198;-5.164;3;1;0;1440
2;5.573;7.114;3;1;0;1440
3;-6.614;0.072;3;1;0;1440
4;-7.374;-1.107;3;1;0;1440
5;-9.251;8.321;3;1;0;1440
6;6.498;-6.036;3;1;0;1440
7;0.861;6.903;3;1;0;1440
8;3.904;-5.261;3;1;0;1440
9;7.976;-9.000;3;1;276;291
10;-2.610;0.039;3;1;32;47
11;4.487;7.142;3;1;115;130
12;8.938;-4.388;3;1;14;29
13;-4.172;-9.096;3;1;198;213
14;7.835;-9.269;3;1;160;175
15;2.792;-7.944;3;1;180;195
16;5.212;9.271;3;1;366;381
17;6.687;6.731;3;-1;402;417
18;-2.192;-9.210;3;-1;322;337
19;-1.061;8.752;3;-1;179;194
20;6.883;0.882;3;-1;138;153
21;5.586;-1.554;3;-1;82;97
22;-9.865;1.398;3;-1;49;64
23;-9.800;5.697;3;-1;400;415
24;1.271;1.018;3;-1;298;313
25;4.404;-1.952;3;-1;0;1440
26;0.673;6.283;3;-1;0;1440
27;7.032;2.808;3;-1;0;1440
28;-0.694;-7.098;3;-1;0;1440
29;3.763;-7.269;3;-1;0;1440
30;6.634;-7.426;3;-1;0;1440
31;-9.450;3.792;3;-1;0;1440
32;-8.819;-4.749;3;-1;0;1440
33;0.000;0.000;0;0;0;480
```

Figura 7.16 Formato archivo de entrada Branch-and-Cut para implementación en Java

Dentro de los archivos de salida, uno corresponde a datos sobre los parámetros de implementación junto con los resultados de tiempo y mejor evaluación, y el otro contiene información sobre algunas de las mejores soluciones encontradas. Uno de ellos es el que se visualiza en la Figura 7.17.

```
LOG DE EJECUCION DE GA:

INFORMACION DE PARAMETROS CONFIGURADOS

1) PARAMETROS DEL GA
Tamaño de la población: 100
Número máximo de generaciones: 1000
Probabilidad de recombinación: 0,7
Probabilidad de mutación en rutas: 0,1
Probabilidad de mutación en clusters: 0,01
Tamaño del torneo: 2

1) PARAMETROS DE FUNCION DARP
Factor del tiempo de duración de las rutas: 1
Factor del tiempo de slacks: 1
Factor de la cantidad de vehículos: 1
Factor del exceso en el tiempo de transporte de clientes: 1
Factor del tiempo de espera de clientes: 1
Factor del tiempo de viaje de clientes: 1

3) OPERADORES UTILIZADOS
Operador recombinación: Recombinación LLGA
Operador mutación en rutas: Mutación de Rutas LLGA
Operador mutación en clusters: Mutación de Clusters LLGA
Generador de población inicial: Generador de pob. inicial para modelo LLGA.
Operador de selección: Selección basada en Torneo

4) CARACTERISTICAS DE LOS DATOS DE ENTRADA
Cantidad de clientes: 16
Cantidad máxima de vehículos: 2
Carga máxima de vehículos: 3
Tiempo máximo de transporte de pasajeros: 00:30:00

(13-10-2010 22:59:15) Comienzo de ejecución del algoritmo.
(13-10-2010 22:59:15) Comienza la generación de la población inicial.
(13-10-2010 22:59:15) Termina la generación de la población inicial. Demora: 00:00:00.1130064
(13-10-2010 22:59:15) Se inicia el avance de generaciones.
(13-10-2010 22:59:20) Fin de ejecución del algoritmo. Demora total: 00:00:04.4162525
```

Figura 7.17 Archivo de salida

# 8 Resultados Experimentales

## 8.1 Diseño de las pruebas

Para ejecutar las pruebas se tomará en cuenta los archivos descritos en el capítulo anterior. Este conjunto de datos de entrada se ha utilizado en ciertos estudios que se han desarrollado para la resolución del DARPTW y se dividen en los dos subconjuntos que se mostraron anteriormente: uno para un algoritmo de Branch-and-Cut y el otro para un algoritmo de búsqueda Tabú.

En una primera etapa se realizará un conjunto de pruebas, comparando el algoritmo implementado en C# y el implementado en Java, para luego incluir las pruebas realizadas en el sistema multiagente. Es importante como un primer acercamiento analizar las diferencias, ya sea en tiempos o resultados en general, de la ejecución del algoritmo en el código original (desarrollado en [18]) y en el código migrado a lenguaje de programación Java, ya que como se mencionó en la sección 6.1.2 el código no es migrado con características idénticas en los dos lenguajes y se espera que los resultados obtenidos no se desvíen de los originales.

Como el presente trabajo supone una continuación de la investigación ya realizada en [18], se utilizarán los mismos archivos de prueba con la misma división de datos que en ella se propuso e igual valor de parámetros para cada conjunto de datos. Lo anterior se muestra en la Tabla 8.2 y 8.3, y en la Tabla 8.1 se encuentran las características de cada archivo utilizado.

| | $\|K\|$ | $n$ | $T$ | $Q$ | $MRT$ |
|---|---|---|---|---|---|
| pr01 | 3 | 24 | 480 | 6 | 90 |
| pr02 | 5 | 48 | 480 | 6 | 90 |
| pr03 | 7 | 72 | 480 | 6 | 90 |
| pr05 | 11 | 120 | 480 | 6 | 90 |
| pr11 | 3 | 24 | 480 | 6 | 90 |
| pr12 | 5 | 48 | 480 | 6 | 90 |
| pr15 | 11 | 120 | 480 | 6 | 90 |
| pr16 | 13 | 144 | 480 | 6 | 90 |
| pr17 | 4 | 36 | 480 | 6 | 90 |
| pr19 | 8 | 108 | 480 | 6 | 90 |
| $\|K\|$ **:** Cantidad máxima de vehículos | | | | | |
| $n$ **:** Cantidad de clientes | | | | | |
| $T$ **:** Duración máxima de la ruta | | | | | |
| $Q$ **:** Capacidad del vehículo | | | | | |
| $MRT$ **:** Tiempo máximo de transporte de pasajeros | | | | | |

Tabla 8.1 Archivos utilizados y sus características

| Conjunto de datos | Archivo de entrada |
|---|---|
| Pequeño | pr01<br>pr02<br>pr11<br>pr12<br>pr17 |
| Mediano | pr03<br>pr05<br>pr15<br>pr19 |
| Grande | pr16 |

Tabla 8.2 Archivos utilizados en cada conjunto de datos

| **Parámetros DARP** | **Valor** |
|---|---|
| Factor del tiempo de duración de las rutas | 8 |
| Factor del tiempo de slacks | 1 |
| Factor de la cantidad de vehículos | 0 |
| Factor del exceso en el tiempo de transporte de clientes | 2 |
| Factor del tiempo de espera de clientes | 0 |
| Factor del tiempo de viaje de clientes | 4 |
| **Parámetros GA** | |
| Tamaño de la población | 100 |
| Número de generaciones máximas de ejecución | 15000 |
| Tamaño del torneo | 2 |
| ***Conjunto de datos pequeño** | |
| Probabilidad de recombinación | 0.45 |
| Probabilidad de mutación en clusters | 0.005 |
| Probabilidad de mutación en rutas | 0.075 |
| ***Conjunto de datos mediano** | |
| Probabilidad de recombinación | 0.35 |
| Probabilidad de mutación en clusters | 0.075 |
| Probabilidad de mutación en rutas | 0.075 |
| ***Conjunto de datos grande** | |
| Probabilidad de recombinación | 0.75 |
| Probabilidad de mutación en clusters | 0.0025 |
| Probabilidad de mutación en rutas | 0.025 |

Tabla 8.3 Parámetros utilizados

## 8.2 Pruebas y resultados

En el proceso de migración del código de C# a Java se utilizaron los archivos usados para el algoritmo de Branch-and-Cut, ya que tienen un tiempo de ejecución mucho menor y el único objetivo en esta etapa consistía en hacer comparaciones entre los dos lenguajes para comprobar que la migración del código se estaba haciendo de manera correcta. Además, se utilizó un máximo de 1000 generaciones, ya que era lo suficiente para verificar un buen funcionamiento.

Pasando a las siguientes pruebas, se realizaron una serie de ejecuciones a las tres implementaciones. La idea principal de esto fue, primero analizar el resultado del cambio de un lenguaje a otro, ya que en este caso implicó desarrollar en Java funcionalidades ya existentes en librerías pertenecientes a C#, para después revisar los resultados referentes a lo que representa la distribución del algoritmo.

Para el caso del algoritmo distribuido se simuló la ejecución en un solo computador. Esto luego de hacer pruebas en tres computadores como esclavos y uno como maestro, donde los resultados en cuanto a CPU fueron deficientes. Los pasos de mensaje se realizaban en tiempos muy altos (10 segundos aproximadamente) y, a pesar de que los procesos a realizar se desarrollaran en un tiempo considerablemente corto, lo antes mencionado producía tiempos de procesamiento total demasiado altos, sobre todo tomando en cuenta que se trabajaría con 15.000 generaciones. Además se implementó en el código el códec Bit-efficient para intentar mejorar los tiempos en el paso de mensajes entre un computador y otro, pero los resultados no fueron los esperados.

A continuación se muestran tres tablas (Tabla 8.4, 8.5 y 8.6) que contienen las mejores soluciones en términos de evaluación para nueve conjuntos de datos procesados, incluyendo además el tiempo de viaje de los vehículos (Travel Time), tiempo de duración de los slacks en la ruta (Slack Time), exceso en el tiempo de viaje de los clientes (Ex. Ride Time), tiempo de espera de los clientes (Wait Time), tiempo de viaje de los clientes (Ride Time) y el tiempo de CPU utilizado.

| C# | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Eval.** | **Travel Time** | **Slack Time** | **Ex.Ride Time** | **Ride Time** | **Wait Time** | **CPU Time** |
| **pr01** | 10366,9 | 959,201 | 47,958 | 140,953 | 590,866 | 1619,924 | 82,171 |
| **pr02** | 19009,8 | 1830,416 | 325,313 | 176,899 | 921,833 | 3264,762 | 169,306 |
| **pr11** | 9039,9 | 905,039 | 0 | 0 | 449,913 | 2316,438 | 84,655 |
| **pr12** | 15199,7 | 1524,435 | 24,543 | 0 | 744,933 | 4692,315 | 186,963 |
| **pr17** | 12290,8 | 1230,157 | 0 | 0 | 612,400 | 3673,944 | 140,703 |
| **pr03** | 31500,0 | 2898,739 | 139,659 | 442,582 | 1821,324 | 4709,513 | 1738,082 |
| **pr05** | 52413,9 | 4204,119 | 307,226 | 1790,245 | 3723,305 | 8016,340 | 4159,955 |
| **pr15** | 52544,7 | 4343,678 | 462,929 | 1600,022 | 3533,081 | 11044,514 | 4725,635 |
| **pr19** | 44187,5 | 3517,938 | 142,387 | 1524,950 | 3212,941 | 10052,648 | 6664,536 |

Tabla 8.4 Mejores evaluaciones para la ejecución en C#

| Java | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Eval.** | **Travel Time** | **Slack Time** | **Ex.Ride Time** | **Ride Time** | **Wait Time** | **CPU Time** |
| **pr01** | 10466 | 950 | 36 | 171 | 622 | 1363 | 62,097 |
| **pr02** | 18945 | 1826 | 323 | 173 | 917 | 2870 | 202,053 |
| **pr11** | 9016 | 903 | 0 | 0 | 448 | 2279 | 70,971 |
| **pr12** | 15020 | 1506 | 0 | 0 | 743 | 4173 | 227,876 |
| **pr17** | 12248 | 1226 | 0 | 0 | 610 | 3015 | 155,668 |
| **pr03** | 31361 | 2797 | 35 | 573 | 1951 | 3864 | 1978,89 |
| **pr05** | 51683 | 4293 | 409 | 1533 | 3466 | 5995 | 5071,583 |
| **pr15** | 51185 | 4092 | 217 | 1748 | 3684 | 10286 | 5338,016 |
| **pr19** | 43300 | 3490 | 118 | 1419 | 3106 | 8952 | 8099,751 |

Tabla 8.5 Mejores evaluaciones para la ejecución en Java

| MAS | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **Eval.** | **Travel Time** | **Slack Time** | **Ex.Ride Time** | **Ride Time** | **Wait Time** | **CPU Time** |
| **pr01** | 10773 | 1070 | 157 | 44 | 492 | 1310 | 2288.462 |
| **pr02** | 18884 | 1828 | 328 | 160 | 903 | 2940 | 4160,898 |
| **pr11** | 9178 | 921 | 18 | 0 | 448 | 2191 | 2288,841 |
| **pr12** | 14936 | 1496 | 0 | 0 | 472 | 4409 | 4282,563 |
| **pr17** | 12232 | 1224 | 0 | 0 | 610 | 3326 | 3292,222 |
| **pr03** | 28824 | 2817 | 64 | 120 | 1496 | 3864 | 6109,837 |
| **pr05** | 42032 | 3902 | 18 | 511 | 2444 | 6634 | 10337,267 |
| **pr15** | 42589 | 3942 | 53 | 546 | 2477 | 6789 | 10264,536 |
| **pr19** | 35678 | 3383 | 0 | 311 | 1998 | 9910 | 10076,054 |

Tabla 8.6 Mejores evaluaciones para la ejecución del MAS

Como se puede observar en las Tablas 8.4, 8.5 y 8.6, el tiempo de CPU utilizado para la ejecución del algoritmo varía parcialmente en los set de datos ejecutados en el código migrado a Java, y se eleva considerablemente para todos los set de datos ejecutados en el sistema multiagente. No existe una diferencia sustancial en los resultados generales de los tiempos referentes al problema en sí, sin embargo la mayoría de ellos son mejores en las implementaciones en Java y en mayor medida en el sistema multiagente.

Observando lo obtenido con el sistema multiagente y comparándolo con las otras dos implementaciones, los resultados de la evaluación para cada set de datos reflejan soluciones con mejores tiempos, sobre todo en lo que se refiere al tiempo de viaje y exceso en el tiempo de viaje de los clientes. Sin tomar en cuenta la implementación en Java no distribuida, como

un punto medio entre las otras dos, si se comparan los resultados conseguidos por el código con que se comenzó esta investigación (en C#) con el que ésta propone (basado en un sistema distribuido a través de agentes), en términos de soluciones ligadas directamente al problema, las entregadas por el sistema multiagente fueron mejores, todo lo contrario a lo sucedido con el tiempo de ejecución para todas las generaciones, donde el programa implementado en C# superó en gran medida al sistema distribuido.

Los gráficos de las Figuras 8.1 y 8.2 muestran los resultados del set de datos pr05, a modo de comparación para el algoritmo ejecutado en C# y en el sistema multiagente.

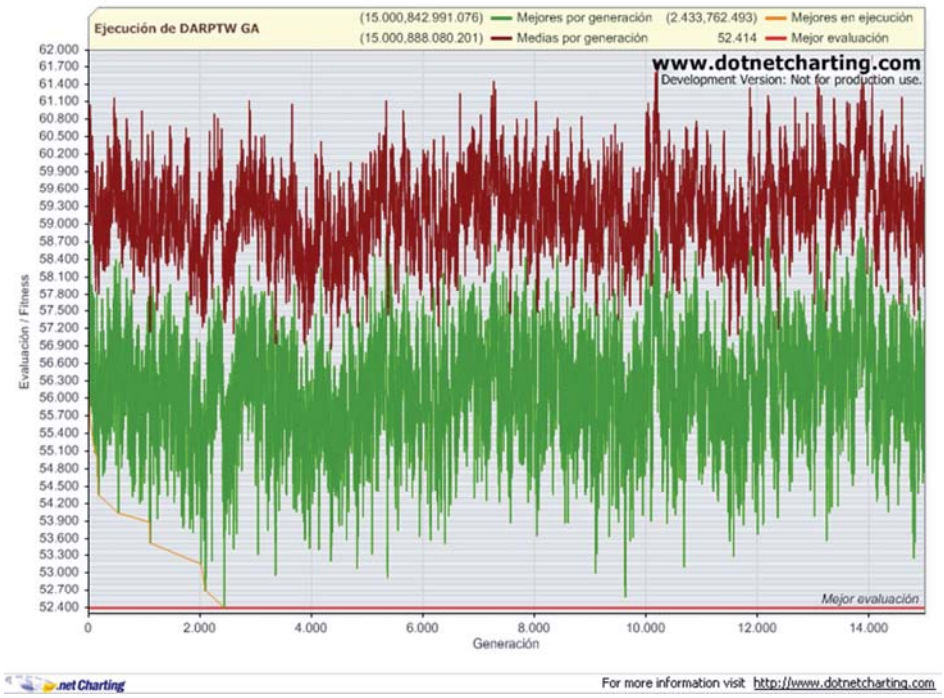


Figura 8.1 Ejecución en C# para el conjunto de datos pr05

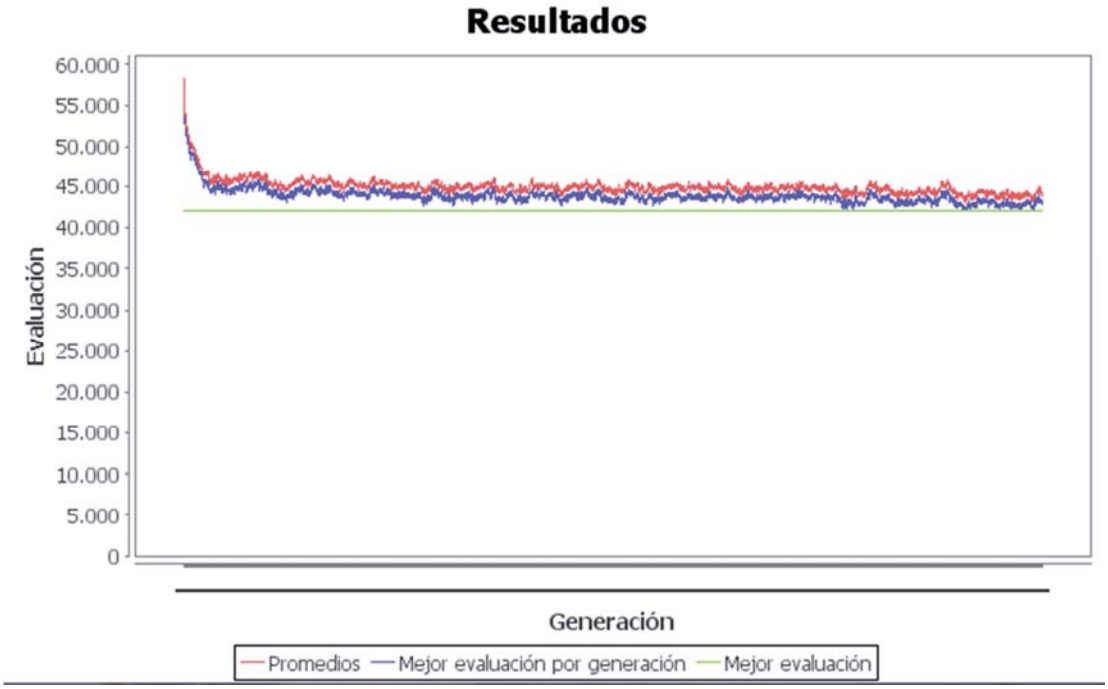


Figura 8.2 Ejecución del sistema multiagente para el conjunto de datos pr05

# 9 Conclusiones y Trabajo Futuro

La computación evolutiva en general, y los algoritmos genéticos en particular, se han convertido en una de las áreas más activas de la Inteligencia Artificial. Sin embargo, el diseño e implementación de algoritmos evolutivos multiobjetivo paralelos es un problema complejo. Como se mencionó, en lo que al resultado de esta investigación se refiere, no existe trabajo alguno relacionado con algoritmos evolutivos para optimización multiobjetivo y sistemas multiagente; por tanto, este trabajo supone la primera tentativa en este sentido.

Por otra parte, se tomó la decisión de dar énfasis al modelo LLGA que obtuvo mejores resultados en términos de tiempo en la investigación anterior. Además, se ha definido el modelo de dominio de agentes y un esquema de distribución basado en maestro-esclavo, el cual ha sido modificado para aprovechar de mejor manera la labor que puede cumplir el agente esclavo dentro de un MAS.

Se desarrolló un prototipo para el sistema propuesto. Como punto de partida se consideró la migración del solver basado en algoritmos genéticos implementado en C#, al lenguaje de programación Java mediante la utilización del IDE Eclipse, para luego trabajar en la implementación del MAS sobre la plataforma JADE. El análisis principal que se hizo para decidir sobre la distribución del algoritmo en los distintos agentes fue de acuerdo a las operaciones que se pensaba utilizarían más tiempo de ejecución, por lo que se optó por la distribución maestro-esclavo explicada en el Capítulo 6, donde los agentes esclavos, a diferencia de una distribución maestro-esclavo normal, realizan las operaciones de cruzamiento y mutación.

En término de los tiempos asociados al problema, el sistema implementado obtuvo mejores resultados que las otras dos implementaciones. Sin embargo, en lo que refiere a tiempo de CPU, la implementación distribuida resulta con deficiencias. Tanto en las pruebas simuladas en sólo una máquina, como en las distribuidas en distintos computadores arrojaron tiempos de CPU muy superiores a las versiones no paralelizadas. La gran diferencia observada en este sentido, se puede deber principalmente a la lógica utilizada en la resolución del problema. La idea desde un comienzo fue conservar dicha lógica, utilizada por el trabajo en que se basó la presente investigación, para así tener un referente de comparación en cuanto a resultados y además dedicar principal énfasis en lo que era el problema distribuido y no en desarrollar una nueva lógica para la resolución del problema, lo que hubiese conllevado un mayor tiempo de investigación y trabajo. Lo anterior produjo que para la primera etapa de migración hubiesen ciertas funcionalidades pertenecientes a las librerías de C# que debieron ser implementadas en Java, ya que no existen como propias del lenguaje y no se encuentran en ninguna de sus librerías. Un claro ejemplo de lo que aplicó mayor complejidad al código en lenguaje Java fue el uso de delegados. Además, en la etapa de distribución del algoritmo ya existían objetos muy complejos, los cuales debían ser pasados como mensajes de un agente a otro y que, aunque se trataron de simplificar sólo manteniendo la información específicamente necesaria para cada agente, implicaban un tiempo de paso de mensajes elevado. Incluso se implementó en el código el códec Bit-efficient para revisar si en los resultados ayudaba en algo a disminuir este tiempo en el paso de los mensajes, pero no se llegó a ninguna mejora. Esta es una de las principales razones de que se utilizara la simulación en un solo equipo de la ejecución del algoritmo distribuido.

Debido a los resultados obtenidos, como trabajo futuro, se hace necesario obtener una mejora en la distribución del algoritmo principalmente para disminuir el tiempo de CPU utilizado. Como se dijo anteriormente en esto influye bastante la lógica utilizada en la implementación original, ya que el código utiliza demasiadas funcionalidades complejas de distribuir y objetos que no pueden ser simplificados para que el envío de mensajes se desarrolle en un tiempo menor. Por lo que se debería pensar en un planteamiento de una nueva resolución del problema con una lógica que se alineara más al lenguaje de programación Java, y que fuese consecuente a una implementación distribuida de éste en un sistema multiagente.

# 10 Referencias


[1] Mitchell, M.: An Introduction to Genetic Algorithms. University of Michigan Press, 1996.
[2] Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning, A. Wesley, 1989.
[3] Koza, J. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Mass. The MIT Press, 1992.
[4] K.A. De Jong (1975). An analysis of the behaviour of a class of genetic adaptive systems. Tesis doctoral, University of Michigan.
[5] Tongchim, Shisanu.: Coarse-Grained Parallel Genetic Algorithm for Solving the Timetable Problem. Informe técnico, 1998.
[6] Cantú-Paz, E.: A Survey of Parallel Genetics Algorithms, Calculateurs Paralleles, Reseaux et Systems Repartis, 1998.
[7] Sarma J., Jong K. D.: An analysis of the effects of neighborhood size and shape on local selection algorithms. In Parallel Problem Solving from Nature IV, p. 236–244, Springer-Verlag (Berlin), 1996.
[8] Schwehm M.: Implementation of genetic algorithms on various interconnection networks. In VALERO M., ONATE E., JANE M., LARRIBA J. L. , SUAREZ B., Eds., Parallel Computing and Transputer Applications, p. 195–203, IOS Press (Amsterdam), 1992.
[9] Bianchini R., Brown C. M.: Parallel genetic algorithms on distributed-memory architectures. In ATKINS S., WAGNER A. S., Eds., Transputer Research and Applications 6, p. 67–82, IOS Press (Amsterdam), 1993.
[10] Russell S.: Inteligencia Artificial: un enfoque moderno. Prentice - Hall. México, 1996.
[11] Michael Wooldridge M. y Jennings N. R.: Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, vol. 10(2) pp. 115-152, 1995.
[12] Haddadi, A. y Sundermeyer K.: Belief-Desire-Intention Agent Architectures. Wiley-Interscience Publication, 1996.
[13] Brooks, R.A.: Intelligence without Representation, Artificial Intelligence, 47, 139-159, 1991.
[14] Tulio José Marchetti, Alejandro Javier García, Una Propuesta de Metodología de Desarrollo de Sistemas Multiagente. Universidad Nacional del Sur, Bahía Blanca. 2004.
[15] Burrafato, P., and Cossentino, M.: Designing a multiagent solution for a bookstore with the passi methodologyIn 4th Int. Bi-Conference Workshop on AgentOriented Information Systems (AOIS-2002).
[16] Cameron Hughes and Tracey Hughes. Parallel and Distributed Programming Using C++. Addison Wesley, Aug. 2003.
[17] Cordeau, J.-F., A Branch-and-Cut Algorithm for the Dial-a-Ride Problem, Operations Research 54, 573-586. Canada Research Chair in Distribution Management, HEC Montréal 3000, Canada. 2003.
[18] Cubillos C., Urra E., Rodríguez N.: Application of Genetic Algorithms for the DARPTW Problem, Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844 Vol. IV (2009), No. 2, pp. 127-136.
[19] Jørgensen, R. M.: Dial-a-Ride. Doctor's thesis, Technical University of Denmark, 2002.
[20] Savelsbergh, M.W.P.: The General Pickup and Delivery Problem. Transportation Science, Vol. 29, pp 17-29, 1995.

[21] Bergvinsdottir, K. B.; Larsen, J.; Jørgensen, R. M.: Solving the Dial-a-Ride Problem using Genetic algorithms. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2004.
[22] A. Grama and V. Kumar, State of the Art in Parallel Search Techniques for Discrete Optimization, IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pp. 28-35, 1999.
[23] UEA CALMA Group, CALMA Project Report 2.4: Parallelism in Combinatorial Optimisation, Tech. Rep., School of Information Systems, University of East Anglia, Norwich, UK, September 18 1995.
[24] A. Migdalas, P. M. Pardalos, and S. Story, Parallel Computing in Optimization (Applied Optimization, Vol. 7), Kluwer Academic Publishers, 1997.
[25] Gendron and T. G. Crainic, Parallel Branch and Bound Algorithms: Survey and Synthesis, Operations Research, vol. 42, 1994.
[26] Alba E. and Tomassini M., Parallelism and Evolutionary Algorithms, IEEE Transactions on Evolutionary Computation, vol. 6, no. 5, pp. 443-462, 2002.
[27] Alba E. and J.M. Troya, A Survey of Parallel Distributed Genetic Algorithms, Complexity, vol. 4, no. 4, pp. 31-52, 1999.
[28] Coello C.A., Van Veldhuizen D.A., and G.B. Lamont, Evolutionary Algorithms for Solving MultiObjective Problems, Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.
[29] K. Deb, P. Zope, and A. Jain, Distributed Computing of Pareto-Optimal Solutions Using MultiObjective Evolutionary Algorithms, Tech. Rep. 2002008, KanGAL, September 2002.
[30] D.A. Van Veldhuizen, J.B. Zydallis, and G.B. Lamont, \Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms," IEEE Transactions on Evolutionary Computation, vol. 87, no. 2, pp. 144-173, April 2003.
[31] I. Foster and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999.
[32] K. Anstreicher, N. Brixius, J.-P. Goux, and J. Linderoth, Solving Large Quadratic Assignment Problems on Computational Grids, Mathematical Programming, vol. 91, pp. 563-588, 2002.
[33] M.O. Neary and P. Cappello, Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing, in Proc. ACM Java Grande/ISCOPE Conference, November 2002, pp. 56-65.
[34] Y. Tanimura, T. Hiroyasu, M. Miki, and K. Aoi, The System for Evolutionary Computing on the Computational Grid," in Parallel and Distributed Computing and Systems (PDCS 2002), November 2002.
[35] S.J. Wright, Solving Optimization Problems on Computational Grids, Tech. Rep., Mathematics and Computer Science Division, Argonne National Latoratory, Argonne, Ill., November 2000.
[36] Whitley, D.: A genetic algorithm tutorial. Statistics and Computing, Vol. 4, pp. 65-85, 1994.
[37] Zhao T., Man Z., Qi X., A CGS-MSM PGA based on Multi-Agent and its application in solving TSP, College of Fluid Power and Control Engineering, 2008 International Conference on Intelligent Computation Technology and Automation.
[38] Cubillos, C.: MADARP: Multi-Agent Framework for Transportation Systems. Tesis para optar al grado de Dottore di Ricerca in Ingenneria Informatica e dei Sistemi (ciclo XVII), Politecnico di Torino, 2005.

[39] Tolmos P.: Introducción a los algoritmos genéticos y sus aplicaciones. Journal Rect@, vol. Actas_10, 2002.
[40] Figueredo C., Rosario P., Loyo J., Larrazabal G.: Un algoritmo evolutivo paralelo mixto para el TSP. Departamento de computación-FACYT, Universidad de Carabobo, Venezuela.
[41] Gómez J.: Metodologías para el desarrollo de sistemas multi-agente. Revista Iberoamericana de Inteligencia Artificial, año/vol. 7, número 018, Asociación Española para la Inteligencia Artificial, Valencia, España, 2003.
[42] Cordeau, J.; Laporte, G.: A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B: Methodological, Vol. 37, pp. 579-594, 2003.
[43] Finin, T., Labrou, Y., & Mayfield, J.: 1997. KQML as an agent communication language. In Software Agents, ed. J. M. Bradshaw. Menlo Park, Calif.: AAAI Press.