



TECNOLÓGICO NACIONAL DE MÉXICO
Instituto Tecnológico de Ciudad Madero

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
División de Estudios de Posgrado e Investigación



**Control de parámetros en algoritmos metaheurísticos para el
problema de VSP**

OPCIÓN I:
Tesis profesional

Que para obtener el título de:
Maestro en Ciencias de la Computación

Presenta:
I.S.C. Javier Alberto Rangel González

Asesor:
D.C.C. Hectór Joaquín Fraire Huacuja



"2015, Año del Generalísimo José María Morelos y Pavón"

Cd. Madero, Tamps; a **10 de Noviembre de 2015.**

OFICIO No.: U5.270/15
AREA: DIVISIÓN DE ESTUDIOS
DE POSGRADO E INVESTIGACIÓN
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

ING. JAVIER ALBERTO RANGEL GONZÁLEZ
NO. DE CONTROL G07070935
PRESENTE

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su examen de grado de Maestría en Ciencias de la Computación, el cual está integrado por los siguientes catedráticos:

- | | |
|-----------------------|-----------------------------------|
| PRESIDENTE : | DR. JOSÉ ANTONIO MARTÍNEZ FLORES |
| SECRETARIO : | DRA. GUADALUPE CASTILLA VALDEZ |
| VOCAL : | DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA |
| SUPLENTE | DR. JUAN JAVIER GONZÁLEZ BARBOSA |
| DIRECTOR DE TESIS : | DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA |
| CO-DIRECTOR DE TESIS: | DR. JUAN JAVIER GONZÁLEZ BARBOSA |

Se acordó autorizar la impresión de su tesis titulada:

"CONTROL DE PARÁMETROS EN ALGORITMOS METAHEURÍSTICOS PARA EL PROBLEMA DE VSP"

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con Usted el logro de esta meta.

Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

ATENTAMENTE
"POR MI PATRIA Y POR MI BIEN"®

M. P. María Yolanda Chávez Circo
M. P. MARÍA YOLANDA CHÁVEZ CIRCO
JEFA DE LA DIVISIÓN



S.E.P.
DIVISION DE ESTUDIOS
DE POSGRADO E
INVESTIGACION
ITCM

c.c.p.- Archivo
Minuta

MYCHC 'NLCO' jar X

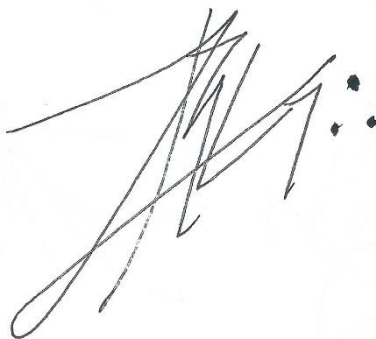


Declaración de Originalidad

Declaro y prometo que este documento de tesis es producto de mi trabajo original y que no infringe los derechos a terceros, tales como derechos de publicación, derechos de autor, patentes y similares.

Además, declaro que en las citas textuales que he incluido (las citas aparecen entre comillas) y en los resúmenes que he realizado de publicaciones ajenas, indico explícitamente los datos de los autores y publicaciones.

Además, en caso de infracción de los derechos de terceros derivados de este documento de tesis acepto la responsabilidad de la información y relevo de ésta área a mi director y codirectores de tesis, así como al Instituto Tecnológico de Ciudad Madero y sus autoridades.

A handwritten signature in black ink, consisting of several overlapping, fluid strokes. The signature is positioned above a horizontal line.

Javier Alberto Rangel González

Contenido

Resumen	2
1. Introducción.....	7
1.1 Objetivos del proyecto	8
1.1.1 Objetivo General	8
1.1.2 Objetivos Específicos.....	8
1.2 Justificación.....	8
1.3 Alcances y Limitaciones	9
2. Estado del Arte.....	10
3. Marco Teórico	14
3.1 Métodos de optimización	14
3.1.1 Problemas de optimización	14
3.1.2 Problema de decisión	15
3.1.3 Complejidad computacional	15
3.1.4 Métodos Exactos	16
3.1.5 Métodos Heurísticos	17
3.1.6 Métodos metaheurísticos	19
3.2 Sistema de colonia de hormigas para TSP (ACS)	20
4. Descripción del problema de Investigación	22
4.1 Problema de Separación de Vértices (VSP)	22
5. Sistemas difusos	25
5.1 Lógica Difusa	25
5.2 Funciones de Membresía	25
5.2.1 Gaussianas.....	26
5.2.2 Triangulares.....	27
5.2.3 Trapezoidales	28
5.3 Reglas de Inferencia	28

5.3.1 Reglas IF-THEN.....	29
5.3.2 Operador AND	29
5.3.3 Operador OR	30
5.3.4 Operador NOT	31
5.4 Diagramas de sistemas lógicos difusos.....	31
5.5 Controladores difusos.....	33
6. Controlador difuso para ACS-TSP, Constructivo Voráz-VB y Exhaustivo-VSP.....	37
6.1 TSP con ACS.....	39
6.2 Vertex Bisection con Algoritmo Constructivo Voráz.....	44
6.3 VSP con Constructivo Voraz.....	51
7. Conclusiones y Trabajos Futuros	57
7.1 Conclusiones.....	57
7.2 Trabajos Futuros	58
Anexo: Productos Científicos	59
Referencias bibliográficas.....	60

Resumen

El problema de separación de Vértices (VSP, por sus siglas en inglés) fue introducido por primera vez en el contexto de encontrar “buenos separadores” para grafos, donde un separador es un subconjunto de vértices o aristas, los cuales al ser removidos separan el grafo en subgrafos desconectados [1]. El VSP es un problema NP-Duro tanto para grafos generales como para grafos estructurados [2].

El VSP tiene aplicaciones en el contexto del diseño de circuitos integrados en una gran escala (VLSI) [3], optimización del posicionamiento de módulos en los circuitos integrados programables (FPGAs) [4, 5, 6], diseño de compiladores [7] y Procesamiento de Lenguaje Natural [8].

Las contribuciones más importantes de este trabajo es un motor difuso tipo 1 adaptable probado en un ACS (Ant Colony System) para el problema de TSP (Traveler Salesman Problem), y en 2 Constructivos Voraces: el primero para el problema de VB (Vertex Bisection) y el segundo para el problema de VSP (Vertex Separation Problem).

Los resultados de la implementación del motor difuso para el problema VB se incorporaron en un artículo titulado “Control Difuso del Parámetro β de una Heurística Constructiva Voráz para el Problema de la Bisección de Vértices de un Grafo” el cual fue publicado en la revista “Research in Computing Science 92”, dicho artículo se encuentra en la sección de Anexos de este documento.

1. Introducción

En la solución de problemas de optimización nos encontramos con muchos algoritmos heurísticos y metaheurísticos que requieren de una cierta afinación y/o control de sus parámetros. Estos parámetros pueden afectar considerablemente el desempeño de los algoritmos.

Existen diferentes maneras de encontrar buenos parámetros para los algoritmos. Sin embargo, es común que requieran de mucho tiempo computacional y estos se mantendrán estáticos durante la ejecución del algoritmo. Como una alternativa a la búsqueda y selección de parámetros estáticos se pueden emplear controladores de parámetros que evalúen el desempeño de los parámetros y modifiquen sus valores en tiempo de ejecución.

Con el objetivo de construir un controlador de parámetros para algoritmos de optimización; se propone, en este trabajo de investigación, el diseño e implementación de un controlador difuso para el control de parámetros. El cual será probado con tres algoritmos diferentes para tres problemas diferentes. En este trabajo se propone el diseño y la implementación de un sistema difuso mediante el cual se pueda controlar los parámetros de una metaheurística dada; resolviendo para este proyecto de tesis un ACS para el problema de TSP, y dos Constructivos Voraces: el primero tipo GRASP para el problema de VB y el segundo para el problema de VSP.

Los conjuntos difusos sirven para realizar una evaluación cualitativa de algunas cantidades físicas. Esto tiene que ver con el razonamiento de que es aproximado y no exacto, es decir, la lógica difusa se maneja con valores entre 0 y 1 a comparación de la lógica booleana que solo maneja el 0 y el 1. La lógica difusa permite evaluar datos difusos, como: valores aproximados, deducciones, así como datos incompletos o ambiguos. Se puede decir que un sistema basado en lógica difusa actúa como lo haría una persona que tuviera que reaccionar ante términos tan imprecisos como “caluroso” o “rápido”.

El Problema de Separación de Vértices (VSP, por sus siglas en inglés) fue introducido por primera vez en el contexto de encontrar “buenos separadores” para grafos, donde un separador es un subconjunto de vértices o aristas, los cuales al ser removidos separan el grafo en subgrafos desconectados [1]. El VSP es un problema NP-Duro tanto para grafos generales como para grafos estructurados [2]. El VSP tiene aplicaciones en el contexto del

diseño de integrados en una gran escala (VLSI) [3], optimización del posicionamiento de módulos en los circuitos integrados programables (FPGAs) [4, 5, 6], diseño de compiladores [7] y Procesamiento de Lenguaje Natural [8].

1.1 Objetivos del proyecto

1.1.1 Objetivo General

Diseñar e implementar un sistema difuso para controlar los parámetros de una solución metaheurística del problema VSP.

1.1.2 Objetivos Específicos

- Revisar el marco teórico de los temas relacionados con el proyecto.
- Revisar el estado del arte de controladores difusos de parámetros de algoritmos metaheurísticos aplicados a problemas relacionados con el VSP.
- Implementación de un sistema difuso para controlar los parámetros de un sistema de colonia de hormigas [16] aplicado a la solución de TSP.
- Diseño e implementación de un sistema difuso para controlar los parámetros de una metaheurística aplicada a la solución del problema VSP.
- Análisis experimental del impacto del controlador difuso en el desempeño del algoritmo correspondiente.

1.2 Justificación

Según a la revisión que se hizo en la literatura, la principal área en la que se ha implementado el uso de los controladores es en el control de parámetros.

Los sistemas difusos presentan una gran ventaja al aplicarse a las metaheurísticas. Ya que permiten controlar comportamiento de los algoritmos durante su ejecución y ajustar algunos parámetros en caso de ser necesario.

1.3 Alcances y Limitaciones

- Este proyecto consiste en diseñar e implementar sistemas difusos para controlar los parámetros de una metaheurística dada.
- El desarrollo de la metaheurística no forma parte de este proyecto.
- Las instancias que serán utilizadas en la evaluación del desempeño de los algoritmos para resolver el problema VSP son las siguientes: GRID, Harwell Boeing, TREE.
- Los lenguajes que se utilizarán para el desarrollo de los algoritmos serán C++ y Java.
- Esto se determina en función del lenguaje en que esté programada la metaheurística a la que se incorpora el controlador.

2. Estado del Arte

Zadeh en 1965 trabajo por primera vez con lógica difusa al mostrar que la lógica difusa era una generalización de la lógica clásica y booleana [22]. De igual manera propuso números difusos como un caso especial de los conjuntos difusos.

Existen varios trabajos reportados en la literatura [23] de optimizar sistemas difusos tipo 2 usando diferentes tipos de algoritmos de optimización de colonia de hormigas (ACO). En Juang et al [24], se presenta un sistema difuso de intervalo tipo 2 con refuerzo auto organizado para configurar el método de optimización de colonia de hormigas (RSOIT2FS-ACO). Los antecedentes en cada regla difusa del RSOIT2FS-ACO usan un conjunto difuso de intervalo tipo 2 para mejorar la robustez del sistema al ruido. Los consecuentes de cada regla difusa fueron diseñados usando las técnicas del ACO. El ACO selecciona los consecuentes de un conjunto de acciones candidatas de acuerdo al rastro de feromonas. El RSOIT2FS-ACO fue aplicado a un control truck-backing. El RSOIT2FS-ACO fue comparado con otros sistemas difusos de refuerzo para verificar su eficiencia y efectividad. Una comparación con sistemas difusos tipo 1 verifico la robustez al ruido de usar un sistema difuso tipo 2.

En el trabajo de Martínez-Marroquín et al [25], se propone el uso de un ACO simple como un método de optimización para los parámetros de las funciones de membresía de un controlador lógico difuso. El uso del ACO permite encontrar el controlador óptimo inteligente para un robot autónomo que se desplaza por medio de llantas. En la implementación del ACO cada controlador difuso de intervalo tipo 2 fue representado como una trayectoria en un grafo. Los resultados de la simulación muestran que el ACO superó a un algoritmo genético en la optimización de controladores lógicos difusos de intervalo tipo 2 para el robot.

En el trabajo de Juang y Hsu [26], se propone un método de diseño para un controlador difuso optimizado con refuerzo de hormigas (RAOFC). Este método fue aplicado a un robot con llantas que se desplaza guiándose por las paredes bajo ambientes de aprendizaje por refuerzo. Las entradas del controlador difuso son sensores sonoros que encuentran el rango, y las salidas son el ángulo de dirección del robot. El antecedente en cada regla difusa usa conjuntos difusos de intervalo tipo 2 para incrementar la robustez del controlador difuso. Fue propuesto un método de agrupamiento difuso de intervalo tipo 2 (AIT2FC) para generar reglas automáticamente. El AIT2FC no solo divide el espacio de entrada sino que también

reduce el número de conjuntos difusos en cada dimensión de la entrada, esto mejora la interpretación del controlador. Los consecuentes de cada regla difusa son designados usando la optimización de colonia de hormigas ayudada por valores Q (QACO). El algoritmo QACO selecciona los consecuentes de un conjunto de acciones candidatas de acuerdo a los rastros de feromona y el valor Q, ambos valores son actualizados usando señales reforzadas. Las simulaciones y experimentos mostraron la eficiencia y efectividad en el RAOFC.

En el trabajo de Castillo et al [27], la aplicación del ACO y la optimización por enjambre de partículas (Particle Swarm Optimization abreviado PSO) para la optimización de un controlador de lógica difusa de intervalo tipo 2 para un robot autónomo que se desplaza por medio de llantas. Los resultados obtenidos por la simulación fueron comparados estadísticamente con los resultados del trabajo previo con algoritmos genéticos para determinar la mejor técnica de optimización para este particular caso del robot. Tanto el PSO como el ACO sobresalieron contra el algoritmo genético para este caso particular. Sin embargo, comparando el ACO y el PSO, los mejores resultados fueron alcanzados por el ACO. En este caso, los autores aseguran que el ACO es más adecuado para este problema en específico.

En el trabajo de Juang y Hsu [28], un nuevo método de aprendizaje reforzado fue propuesto usando el algoritmo de optimización de colonia de hormigas ayudado con valores Q con generación de reglas en línea (Online Rule Generation and Q-value-aided Ant Colony Optimization abreviado ORGQACO) para el diseño del controlador difuso. El controlador difuso está basado en un sistema difuso de intervalo tipo 2. Los antecedentes en el IT2FS usan conjuntos difusos de intervalo tipo 2 para mejorar la robustez del controlador al ruido. Fue propuesto un método de generación de reglas de intervalo tipo 2 en línea para la evolución de la estructura del sistema y particionar el espacio de entrada.

Los parámetros de los consecuentes en el IT2FS fueron diseñados usando el algoritmo de optimización de colonia de hormigas con valores Q y refuerzo local-global. Este algoritmo selecciona los consecuentes de un conjunto de acciones candidatas de acuerdo a los rastros de feromona y valores Q, ambos son actualizados usando señales de refuerzo. El método ORGQACO fue aplicado a los 3 siguientes problemas de control: (1) Control Truckbacking; (2) Control de levitación magnética; y (3) Control de sistemas caóticos. El ORGQACO fue comparado con otros métodos de aprendizaje reforzado para verificar su

eficiencia y efectividad. Las comparaciones con sistemas difusos tipo 1 verifica la robustez de usar un IT2FS.

En la tabla 2.1 se presenta un resumen de las contribuciones mencionadas anteriormente, donde ACO fue aplicado para optimizar sistemas difusos Tipo-2 en problemas de control. En la tabla 2.1 se muestra que hasta el momento todos los trabajos han sido hechos en el área de controladores lógicos difusos tipo 2 usando diferentes métodos de ACO. La comparación mostrada está basada en el siguiente criterio: nombre de los autores, año de la publicación, número de referencia, dominio del problema, si una comparación con la lógica difusa tipo 1 es presentada, si una comparación con otro método de optimización es presentada, y porqué la lógica difusa tipo 2 fue usada por los autores.

Tabla 2.1. Resumen de las contribuciones del estado del arte.

Autor(es)	Dominio del problema	¿Se hace una comparación con el Tipo-1?	Comparación con otro método de optimización	¿Porque se requiere el Tipo-2 en ese problema?
Juang et al. [24]	Control	Si	No	Incertidumbre en control
Martínez-Marroquin et al. [25]	Control	Si	Si	Incertidumbre en robots móviles
Juang and Hsu [26]	Control	No	No	Incertidumbre en navegación
Castillo et al. [27]	Control	Si	Si	Incertidumbre en control
Juang and Hsu [28]	Control	Si	No	Probar diferentes controladores

Como se puede observar en la tabla, en varios trabajos que se encontraron en la literatura ese hace uso de algoritmos de colonia de hormigas para la optimización de sistemas difusos de tipo 2, pero su aplicación ha sido principalmente en el área de control. Razón por la cual se puede creer que en otras áreas de optimización se pueden llegar a obtener buenos resultados como los obtenidos para problemas del área de control. También se puede observar que para el problema VSP no se encontró ningún trabajo en el cual se proponga una solución haciendo uso de controladores de parámetros difusos de ningún tipo.

3. Marco Teórico

3.1 Métodos de optimización

3.1.1 Problemas de optimización

Se puede decir que un problema de optimización es simplemente un problema en el que hay varias (en general muchas) posibles soluciones y alguna forma clara de comparación entre ellas, de manera que éste existe si y sólo si se dispone un conjunto de soluciones candidatas diferentes que pueden ser comparadas [9].

Desde un punto de vista matemático, un problema de optimización P se puede formular como una 3-tupla $P = (f, SS, F)$, definida como sigue:

$$P = \begin{cases} \text{opt: } f(x), & \text{Función Objetivo} \\ \text{sujeto a} & \\ x \in F \subset SS & \text{Restricciones} \end{cases}$$

Donde f es la función a optimizar (maximizar o minimizar), F es el conjunto de soluciones factibles y SS es el espacio de soluciones.

Los problemas de optimización se dividen en dos categorías [10]: aquéllos en los que la solución está codificada mediante valores reales y aquéllos cuyas soluciones están codificadas por valores enteros. Dentro de la segunda categoría se encuentran un tipo particular de problemas denominados problemas de optimización combinatoria.

Un Problema de Optimización Combinatoria consiste en encontrar entre un conjunto finito de soluciones potenciales (llamado espacio de búsqueda) la o las mejores soluciones que verifiquen un criterio particular. Para resolver los problemas de optimización combinatoria se pueden usar métodos aproximados o exactos. Los algoritmos exactos permiten explorar sistemáticamente un conjunto de soluciones potenciales por lo que siempre proporcionan la mejor solución (óptima) al problema en cuestión. Los métodos aproximados a su vez se puede dividir en dos categorías: algoritmos a la medida y algoritmos generales. Los algoritmos a la medida son aquellos que fueron diseñados para resolver un problema específico. Los algoritmos generales, en cambio, son fácilmente adaptables a la solución de una amplia variedad de problemas de optimización [9].

Existen tres conceptos básicos que se pueden encontrar en la resolución algorítmica de problemas de optimización combinatoria. Independientemente de la técnica que se utilice, se debe especificar:

- Representación: se encarga de codificar las soluciones factibles para su manipulación. Determina el tamaño del espacio de búsqueda (SS) de cada problema.
- Objetivo: es un predicado matemático que expresa la tarea que se tiene que realizar.
- Función de evaluación: permite asociar a cada solución factible un valor que determina su calidad.

3.1.2 Problema de decisión

Garey *et al* [11] definen un problema de decisión Π como un conjunto D_Π de instancias y un subconjunto $Y_\Pi \subseteq D_\Pi$ de si-instancias. Donde una instancia pertenece a D_Π si y solo si puede obtenerse a partir de una instancia genérica mediante la sustitución de objetos particulares de los tipos específicos para todos los componentes genéricos, así mismo una instancia pertenece a Y_Π si y solo si la respuesta a la cuestión del problema es sí para esa instancia.

3.1.3 Complejidad computacional

La teoría de la complejidad computacional es la rama de la teoría de la computación que estudia, de manera teórica, la complejidad inherente a la resolución de un problema computable. Los recursos comúnmente estudiados son el tiempo (mediante una aproximación al número y tipo de pasos de ejecución de un algoritmo para resolver un problema) y el espacio (mediante una aproximación a la cantidad de memoria utilizada para resolver un problema). La mayor parte de los problemas en teoría de la complejidad tienen que ver con los problemas de decisión.

Los problemas de decisión se clasifican en conjuntos de complejidad comparable llamados clases de complejidad [9] [11].

- **Clase P.** Es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina determinista en tiempo polinomial.

- **Clase NP.** Es el conjunto de los problemas de decisión que pueden ser resueltos por una máquina no determinista en tiempo polinomial.
- **Clase NP-completo,** Son aquéllos problemas que son al menos tan difíciles como todos los que se encuentran en NP. Y ellos mismos pertenecen a NP. Se cree que este tipo de problemas es un sub conjunto de NP.
- **Clase NP-duro.** Los problemas NP-duros son al menos tan difíciles como todos los que se encuentran en NP sin embargo, ellos mismos no se encuentran en NP. Una forma de que esto suceda es con la versión de optimización de un problema que se encuentra en la clase NP-Completo. Como un problema de decisión es un caso particular de un problema de optimización entonces los problemas en NP se pueden transformar a la versión de optimización sin embargo como la teoría de la NP completes solo aplica a problemas de decisión entonces un problema de optimización no se puede encontrar dentro de NP

En la figura 2.1 se muestra la relación entre los problemas P, NP, NP-completo y NP-duro.

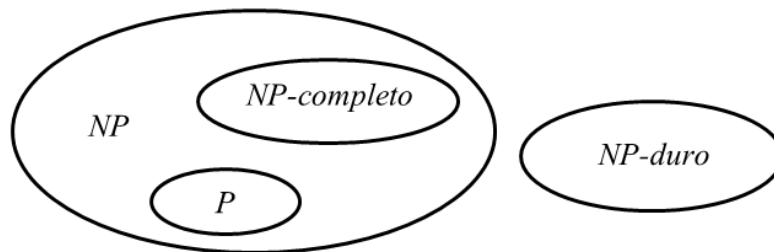


Figura 2.1 Relación entre los problemas P, NP, NP-completo y NP-duro.

3.1.4 Métodos Exactos

Los problemas combinatorios presentan como particularidad que siempre existe un algoritmo exacto que permite obtener la solución óptima, aunque dependiendo del tamaño del problema este podría tomar demasiado tiempo para ser de uso práctico. Esta solución se puede obtener a partir de diferentes métodos; uno de ellos es empleando modelos de programación lineal enteros o mixtos, otro empleando algoritmos de acotamiento del conjunto de soluciones factibles, y otros consisten en la exploración de forma exhaustiva del conjunto de soluciones (enumeración) [12]. Algunas de estas técnicas particulares se muestran a continuación [13]:

- **Branch and bound.** El método *branch and bound* es un enfoque casi enumerativo para resolver una variedad de problemas combinatorios. Esto es bastante eficiente para problemas de tamaño modesto, y la metodología general forma una parte importante del conjunto de métodos de solución para la clase general de problemas de programación lineal entera. Su idea básica es particionar un problema dado en un número de subproblemas (*branching*). Propone establecer subproblemas que son más fáciles de resolver que el problema original, por su tamaño menor o estructura susceptible [12].
- **Enumeración Implícita.** Esta técnica es usualmente aplicada a problemas de programación entera cero-uno. Es similar al *branch and bound*; sin embargo, las reglas de ramificación, restricción y acotamiento son simplificadas y refinadas porque cada variable entera puede tomar solo valores cero o uno. Los problemas de programación entera cero-uno tienen un número finito de puntos factibles, 2^n puntos enteros factibles, donde n es el número de variables cero-uno.
- **Planos de corte.** El objetivo de este método es iterativamente construir el casco convexo en la vecindad de la solución óptima entera. La solución óptima de un problema de programación entera puede ser determinado resolviendo un problema de programación lineal única en la que el casco convexo es usado como la región factible. Esto es porque los puntos de los extremos del casco convexo corresponden a las soluciones enteras. Se hace de una manera sistemática introduciendo restricciones adicionales que cortan porciones de la región factible excluyendo algunos puntos enteros factibles.

3.1.5 Métodos Heurísticos

Para la mayoría de problemas de interés no se conoce un algoritmo exacto con complejidad polinomial que encuentre la solución óptima a dicho problema. Debido a esto se deben utilizar métodos aproximados o heurísticas que permitan obtener una solución de calidad en un tiempo razonable a estos problemas [9]. Una de las definiciones más claras del término heurísticas presentada por Zanakis *et al* [14] es:

“Procedimientos simples, a menudo basados en el sentido común, que se supone que obtendrán una buena solución (no necesariamente óptima) a problemas difíciles de un modo sencillo y rápido”.

Los métodos heurísticos tienen su principal limitación en su incapacidad para escapar de óptimos locales. Esto se debe, fundamentalmente, a que estos algoritmos no utilizan ningún mecanismo que les permita proseguir la búsqueda del óptimo en el caso de quedar atrapados en un óptimo local.

Para solventar este problema, se introducen otros algoritmos de búsqueda más potentes (denominados metaheurísticas) que evitan, en la medida de lo posible, este problema. Este tipo de algoritmos son procedimientos de alto nivel que guían a métodos heurísticos conocidos, evitando que éstos queden atrapados en óptimos locales.

A continuación se muestra una clasificación de las heurísticas de acuerdo a [9]:

1. **Métodos constructivos:** Procedimientos que son capaces de construir una solución a un problema dado. La forma de construir la solución depende de la estrategia seguida. Las más comunes son las siguientes:
 - Estrategia voraz: Partiendo de una semilla, se va construyendo paso a paso una solución factible.
 - Estrategia de descomposición: Se divide sistemáticamente el problema en subproblemas más pequeños.
 - Métodos de reducción: Identifican características que contienen las soluciones buenas conocidas y se asume que la solución óptima también las tendrá.
 - Métodos de manipulación del modelo: Consiste en simplificar el modelo del problema original para obtener una solución al problema simplificado.
2. **Métodos de búsqueda:** Parten de una solución factible dada y a partir de ella intentan mejorarla, a continuación se muestran algunas propuestas.
 - Estrategia de búsqueda local *first*: Parte de una solución factible y la mejora progresivamente. Para ello examina su vecindad y selecciona el primer movimiento que produce una mejora en la solución actual (first improvement).
 - Estrategia de búsqueda local *best*: Parte de una solución factible que la mejora progresivamente. Par ello examina su vecindad y todos los posibles movimientos seleccionando el mejor movimiento de todos los posibles (best improvement).
 - Estrategia aleatorizada: Para una solución factible dada y una vecindad asociada a esa solución, se seleccionan aleatoriamente soluciones vecinas de esa vecindad.

3.1.6 Métodos metaheurísticos

El término metaheurística o meta-heurística fue acuñado por F. Glover en el año 1986 [9]. Glover pretendía definir un procedimiento maestro de alto nivel que guía y modifica otras heurísticas para explorar soluciones más allá de la simple optimalidad local. Una de las definiciones más descriptivas del término metaheurística es la presentada por J.P. Kelly *et al* [15], que se puede enunciar del siguiente modo:

“Las metaheurísticas son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria en los que los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos”.

Existen diferentes clasificaciones en cuanto a metaheurísticas, a continuación se presenta la clasificación con relación a la cantidad de soluciones empleadas [9] [12]:

- **Metaheurísticas trayectoriales.** Se utiliza el término trayectoria porque el proceso de búsqueda que desarrollan estos métodos se caracteriza por una trayectoria en el espacio de soluciones. Es decir, que partiendo de una solución inicial, son capaces de generar un camino o trayectoria en el espacio de búsqueda a través de operaciones de movimiento.
- **Metaheurísticas poblacionales:** Las metaheurísticas basadas en poblaciones o metaheurísticas poblacionales son aquellas que emplean un conjunto de soluciones (población) en cada iteración del algoritmo, en lugar de utilizar una única solución como las metaheurísticas trayectoriales.

3.2 Sistema de colonia de hormigas para TSP (ACS)

El nombre de esta metaheurística [16] viene de su acrónimo en inglés Ant Colony System (ACS), que en castellano se podría traducir como Sistema de Colonia de Hormigas. ACS se deriva de su predecesor llamado Ant System (AS), a continuación se muestra su representación para TSP.

Sea:

- $V = \{a, \dots, z\}$ un conjunto de ciudades,
- $A = \{(r, s) : r, s \in V\}$ un conjunto de aristas,
- y $\delta(r, s) = \delta(s, r)$ un costo asociado con el arista $(r, s) \in A$.

En complemento a la medida del costo $\delta(r, s)$, cada arista (r, s) también tiene un valor que indica que tan deseable es el camino de r a s definido como $\tau(r, s)$, llamada feromona, que se actualiza durante el proceso por las hormigas. Cuando el Ant System es aplicado a instancias simétricas del TSP, $\tau(r, s) = \tau(s, r)$, mientras que cuando son aplicadas a instancias asimétricas es posible que $\tau(r, s) \neq \tau(s, r)$.

El Ant Colony System (ACS) difiere de su predecesor en 3 aspectos principales:

- La regla de transición de estado provee una forma directa de balancear entre la exploración de nuevas aristas, la exploración a priori y el conocimiento acumulado acerca del problema.
- La regla de actualización global es aplicada sólo a las aristas que pertenecen al mejor recorrido por hormigas.
- Mientras que las hormigas construyen una solución, una regla de actualización de la feromona local es aplicada

El Ant Colony System trabaja de la siguiente manera [16]:

m hormigas son posicionadas inicialmente en n ciudades elegidas de acuerdo a una regla de inicialización (por ejemplo, aleatoriamente). Cada hormiga genera un circuito aplicando repetidamente una regla de transición de estado. Mientras construye su circuito, cada hormiga también modifica la cantidad de feromona en las aristas visitadas aplicando la regla de actualización local. Una vez que todas las hormigas han terminado su circuito, la cantidad de feromona en las aristas es modificada nuevamente aplicando la regla de

actualización global. Al igual que en el Ant System, las hormigas son guiadas en construir sus circuitos por la información heurística (preferencia por las aristas pequeñas), y por la información de la feromona: Una arista con alta cantidad de feromona es una opción muy deseable. Las reglas de actualización de la feromona son diseñadas para dar mayor cantidad de feromona a las aristas que deben ser visitadas por las hormigas.

Algoritmo 1: Algoritmo ACS

1. Algoritmo ACS(*Iteraciones*)
 2. InicializaEstructurasDeDatos()
 3. For $i=1 \dots Iteraciones$
 4. PosicionarHormiga(*randomNode*)
 5. While(SolucionesCompletas()==False)
 6. ReglaTransiciónEstado(*Hormiga, Solución*)
 7. ReglaFeromonaLocal(*Hormiga, Solución*)
 8. End
 9. ReglaFeromonaGlobal(*Solución*)
 10. End
 11. Return *MejorSolución*
 12. End
-

4. Descripción del problema de Investigación

4.1 Problema de Separación de Vértices (VSP)

El problema de la Separación de los vértices de un grafo es un **problema de minimización** [21] que consiste en encontrar un subconjunto de los vértices, los cuales al ser removidos separan al grafo en dos sub grafos desconectados.

El contexto de aplicación más importante se encuentra en el diseño de VLSI, para particionar circuitos en subsistemas de menor tamaño, con un mínimo número de componentes en la frontera, optimizando el área y la longitud de cable utilizados.

Definición Formal del VSP

Para poder realizar la definición formal del problema, es necesario primero definir los conceptos básicos referentes a la nomenclatura [2]:

$G = (V, E)$: Representa una instancia del problema, es decir, un grafo no dirigido.

$V(G)$: Representa al conjunto de vértices del grafo G .

$E(G)$: Representa al conjunto de aristas del grafo G .

n : Representa el número total de nodos del grafo, es decir $n = |V(G)|$.

$\varphi(G)$: Representa una ordenación lineal del grafo G , es una función biyectiva $\varphi: V(G) \rightarrow \{1, 2, \dots, n\}$.

$\Phi(G)$: Representa al conjunto de todas las ordenaciones lineales del grafo G , es decir, el espacio de soluciones.

i : Representa un punto de corte de la ordenación lineal, $\forall i = 1, 2, \dots, n - 1$.

$L(i, \varphi, G) = \{u \in V(G) | \varphi(u) \leq i\}$: Representa al conjunto de todos los vértices que quedan a la izquierda del punto de corte i , a este conjunto se le denomina conjunto izquierdo.

$R(i, \varphi, G) = \{u \in V(G) | \varphi(u) > i\}$: Representa al conjunto de todos los vértices que quedan a la derecha del punto de corte i , a este conjunto se le denomina conjunto derecho.

$\delta(i, \varphi, G) = \{u \in L(i, \varphi, G) | \exists v \in R(i, \varphi, G), (u, v) \in E(G)\}$ Representa al conjunto de todos los vértices del conjunto izquierdo que tienen al menos un vértice adyacente en el conjunto derecho. Este conjunto representa la función objetivo de una ordenación lineal φ .

Una manera natural de representar una ordenación lineal φ de un grafo G consiste en alinear sus vértices, asignando al vértice u la posición $\varphi(u)$, tal como se muestra en la figura 2.1.

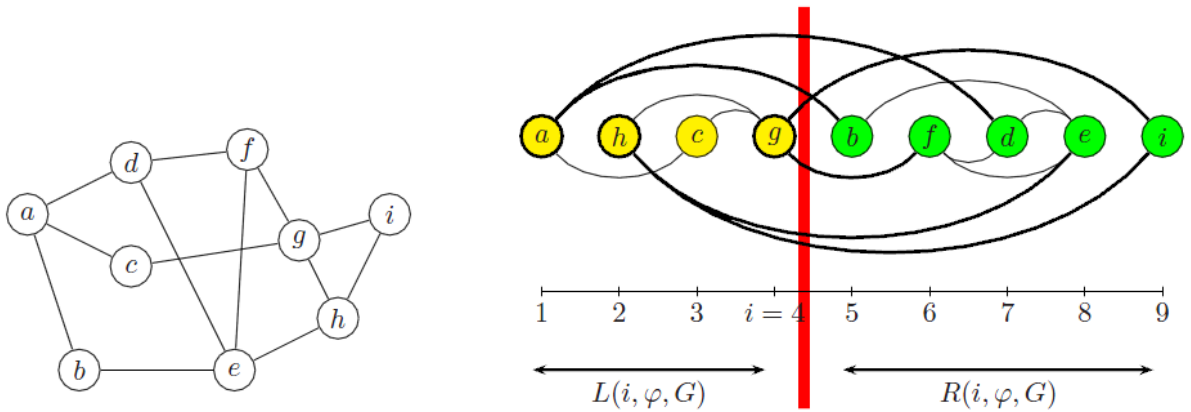


Figura 4.1 Un grafo G y una representación de una ordenación lineal φ de G [2].

Debido a que en una ordenación lineal φ existen múltiples valores objetivo posibles (ya que existen $n - 1$ puntos de corte i), se toma como valor objetivo de la ordenación lineal, el valor máximo de todos los puntos de corte i de φ , es decir:

$$vs(\varphi, G) = \max_{i \in V(G)} (\delta(i, \varphi, G)) \quad (1)$$

El costo de una ordenación lineal (valor objetivo) es una función F que asocia a cada ordenación lineal φ de un grafo G con un entero $F(\varphi, G)$. Sea F el valor objetivo de la ordenación lineal; el problema de optimización de ordenación lineal asociado con F consiste en determinar alguna ordenación lineal $\varphi^* \in \phi(G)$ de un grafo G de entrada, tal que:

$$vs(\varphi^*, G) = \min_{\varphi \in \phi(G)} F(\varphi, G) \quad (2)$$

5. Sistemas difusos

5.1 Lógica Difusa

La lógica difusa es una extensión de la lógica tradicional (booleana) que utiliza conceptos de pertenencia de conjuntos más parecidos a la manera de pensar humana, por ejemplo, los humanos pensamos con diversos grados de verdad, los cuales son difusos (“mucho”, “poco”, “joven”, “adulto”, “adulto mayor”, etc.)[2].

5.2 Funciones de Membresía

Un conjunto difuso, en comparación con un “conjunto clásico” no cuenta con límites discretos y claramente definidos. En la lógica difusa, la verdad de cada declaración se vuelve un grado de pertenencia. Razonando en lógica difusa se le asigna un grado de pertenencia a cada estado, por ejemplo, en la figura 5.1 a los 25 años se le asigna un grado de pertenencia de 0.8 a *joven*, de 0.5 a *adulto* y de 0 a *adulto mayor* aproximadamente.

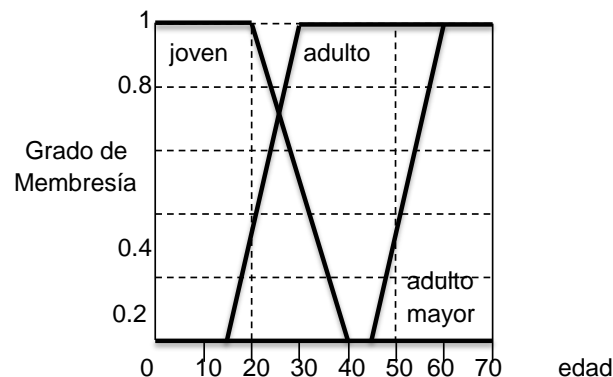


Figura 5.1

Una función de membresía (MF ó μ) es una curva que define como cada punto del espacio de entrada es asignado a un valor de membresía entre 0 y 1.

Un conjunto clásico es definido por una membresía discreta; por ejemplo:

$$A = \{x, \mu_A(x) | x \in X\}$$

Un conjunto difuso es una extensión de un conjunto clásico donde la función de membresía describe el grado de pertenencia. Si X es el universo de búsqueda y sus

elementos son representados por x , entonces un conjunto difuso A en X , es definido como un conjunto de pares ordenados:

$$A = \{x, \mu_A(x) | x \in X\}$$

$\mu_A(x)$ es llamada la función de membresía de x en A .

En las figuras siguientes se muestra una función de membresía Gaussiana, triangular y trapezoidal tipo 1, que está restringida para tomar valores entre 0 y 1 para toda $x \in X$, y es una función bidimensional; este tipo de función de membresía no contiene ninguna incertidumbre; es decir, existe un valor de membresía para cada punto [3]. Para el intervalo tipo 2 la función de membresía cuenta con una huella de incertidumbre; es decir, existe más de un solo valor de membresía para cada punto pero dado que la segunda función de membresía es de intervalo tipo 2, su valor es igual a 1 [4].

5.2.1 Gaussianas

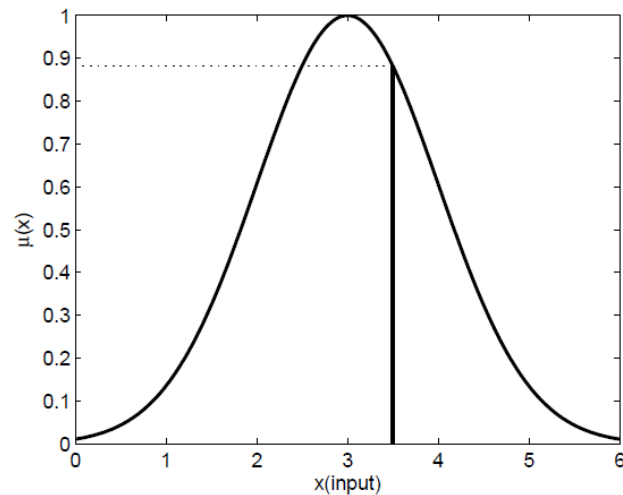


Figura 5.2

La fórmula usada para generar esta función de membresía es:

$$\mu_A(x) = e^{-\frac{x-m^2}{(2k)^2}}$$

Donde k es la desviación estándar y m el centro de la campana.

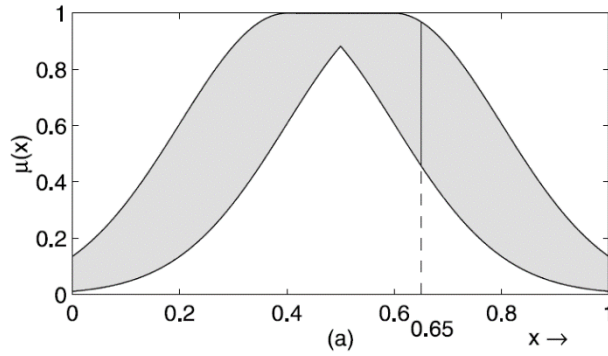


Figura 5.3

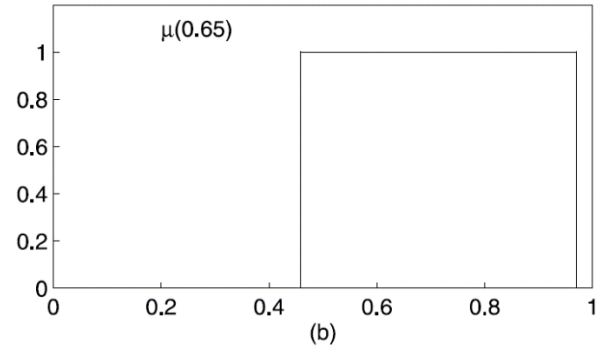


Figura 5.4

5.2.2 Triangulares

Definida mediante el límite inferior a , el superior b y el valor modal m , tal que $a < m < b$. La función no tiene porqué ser simétrica.

$$\mu_A(x) = \begin{cases} 0 & \text{Si } x \leq a \text{ ó } x \geq b \\ \frac{x - a}{m - a} & \text{Si } a < x \leq m \\ \frac{b - x}{b - m} & \text{Si } m < x < b \end{cases}$$

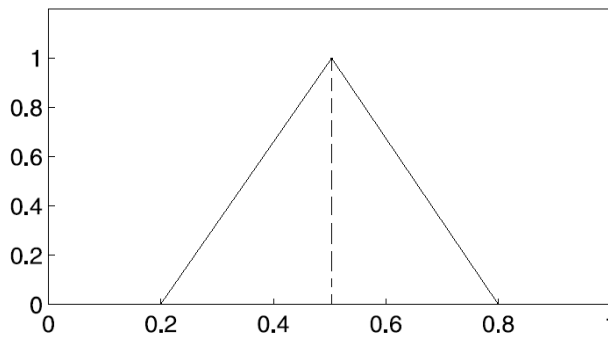


Figura 5.5

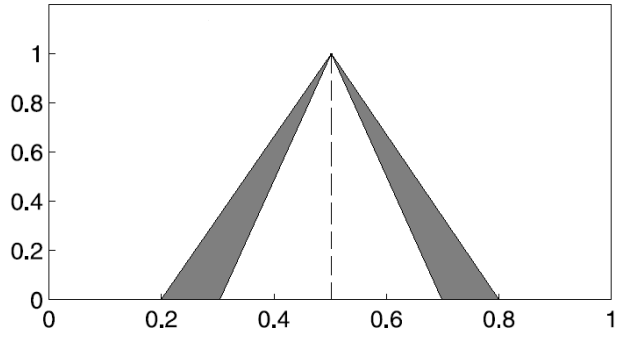


Figura 5.6

5.2.3 Trapezoidales

Definida por sus límites inferior a , superior d , y los límites de soporte inferior b y superior c , tal que $a < b < c < d$. En este caso, si los valores de b y c son iguales, se obtiene una función triangular.

$$\mu_A(x) = \begin{cases} 0 & \text{Si } x < a \text{ ó } x > d \\ \frac{x - a}{b - a} & \text{Si } a \leq x \leq b \\ 1 & \text{Si } b \leq x \leq c \\ \frac{d - x}{d - c} & \text{Si } c \leq x \leq d \end{cases}$$

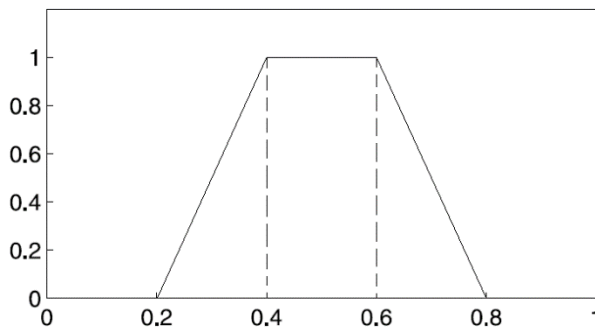


Figura 5.7

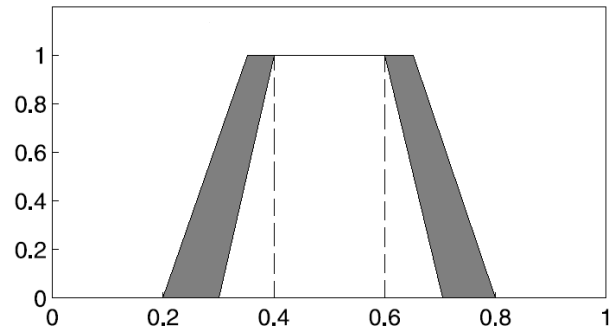


Figura 5.8

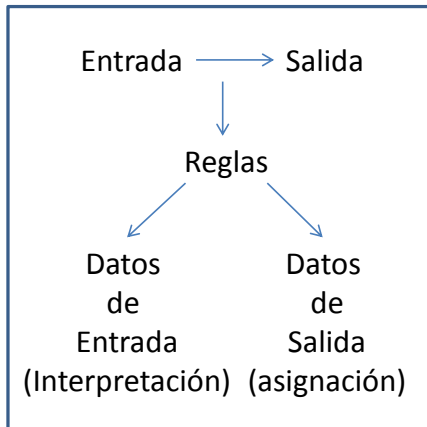
5.3 Reglas de Inferencia

El objetivo de la lógica difusa es asignar un espacio de entrada a un espacio de salida, y el mecanismo primario para lograr esto es una lista de estados *if – then* llamados reglas.

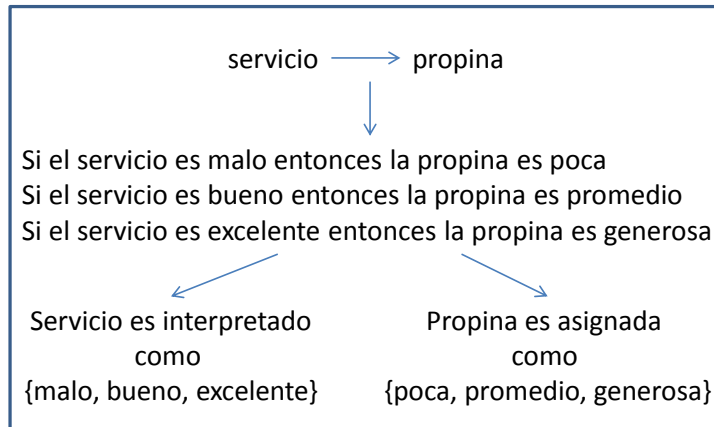
Todas las reglas son evaluadas en paralelo, y no importa el orden de las reglas. Las reglas se refieren a las variables y a los adjetivos que describen a esas variables.

La inferencia difusa es un método que interpreta los valores del vector de entrada y, basado en un conjunto de reglas, asigna valores al vector de salida.

Caso General



Ejemplo



5.3.1 Reglas IF-THEN

Los conjuntos difusos y los operadores difusos son los sujetos y los verbos de la lógica difusa respectivamente. Las reglas IF-THEN son usadas para formular las declaraciones condicionales de la lógica difusa.

Una regla difusa sencilla IF-THEN asume la forma:

If x es A **then** y es B

Donde A y B son valores lingüísticos definidos por un conjunto difuso en los rangos de X y Y respectivamente.

La parte **if** de la regla " x es A " es llamada antecedente o premisa, mientras que la parte **then** de la regla " y es B " es llamada consecuente o conclusión.

5.3.2 Operador AND

La función de membresía de la intersección de 2 conjuntos difusos A y B con funciones de membresía μ_A y μ_B respectivamente, es definida como el mínimo de las 2 funciones de membresía individuales. Esto es llamado el criterio mínimo.

$$\mu_{A \cap B} = \min(\mu_A, \mu_B)$$

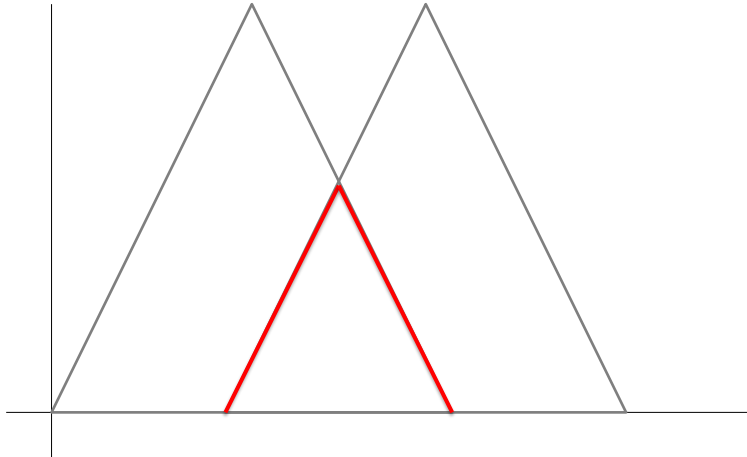


Figura 5.9

La operación Intersección en teoría de conjuntos difusos es equivalente al operador *AND* en algebra booleana.

5.3.3 Operador OR

La función de membresía de la unión de los conjuntos difusos *A* y *B* con funciones de membresía μ_A y μ_B respectivamente, es definida como el máximo de las dos funciones de membresía individuales. Esto es llamado el criterio máximo.

$$\mu_{A \cup B} = \max(\mu_A, \mu_B)$$

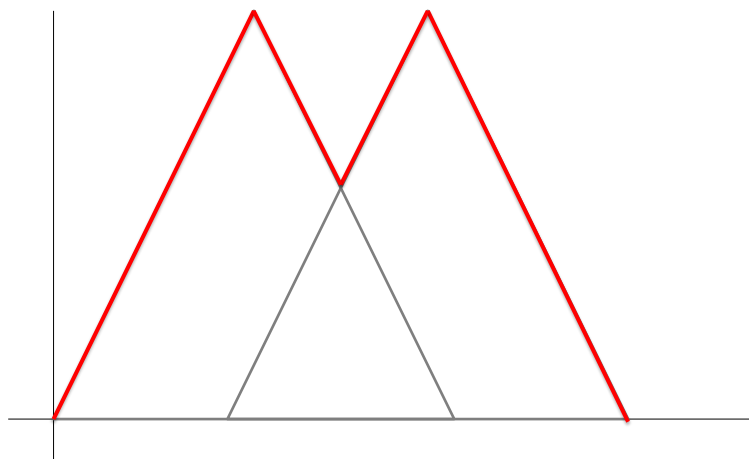


Figura 5.10

La operación Unión en teoría de conjuntos difusos es equivalente al operador *OR* en algebra booleana.

5.3.4 Operador NOT

La función de membresía Complemento de un conjunto difuso A con una función de membresía μ_A es definida como la negación de la función de membresía especificada. Esto es llamado el criterio negado.

$$\mu_{\bar{A}} = 1 - \mu_A$$

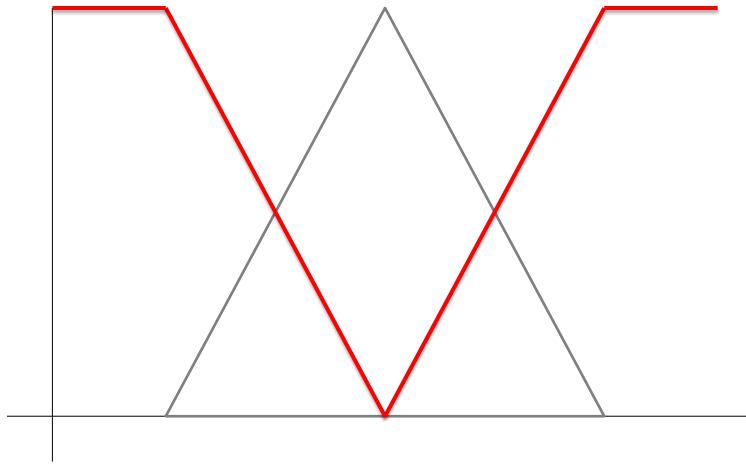


Figura 5.11

La operación Complemento en teoría de conjuntos difusos es equivalente al operador *NOT* en algebra booleana.

5.4 Diagramas de sistemas lógicos difusos

El sistema de inferencia difusa es una estructura computacional muy popular basada en los conceptos de la teoría difusa, en reglas del tipo si-entonces y en métodos de inferencia difusa. Los sistemas de inferencia difusa, se encuentran aplicados a una gran variedad de áreas tales como el control automático, la clasificación de datos, el análisis de decisiones, los sistemas expertos, la predicción de series de tiempo, la robótica y en el reconocimiento de patrones.

A causa de su naturaleza multidisciplinaria, los sistemas de inferencia difusa son conocidos como sistemas experto, modelos difusos, controladores lógicos difusos o simplemente como sistemas difusos.

A continuación se presentan tanto los diagramas, como las especificaciones de cada segmento de los diagramas sistema difuso de tipo 1 (T1FLS) y sistema difuso de tipo 2 (T2FLS) [19]:

- **Fuzzificador:** El fuzzificador mapea las entradas no difusas en los conjuntos difusos que activan el motor de inferencia.
- **Conjunto de reglas:** Las reglas en un T1FLS constan de antecedentes y consecuentes que son representados por un conjunto difuso tipo 1.
- **Inferencia:** El segmento de inferencia asigna entradas difusas a salidas difusas usando las reglas del conjunto de reglas y los operadores como la unión y la intersección.
- **Defuzzificación:** Las salidas del segmento de reducción de tipo son dadas al segmento de defuzzificación. Los conjuntos de tipo reducido son determinados por su punto de los extremos, izquierdo y derecho, el valor defuzzificado es calculado por el promedio de estos puntos.

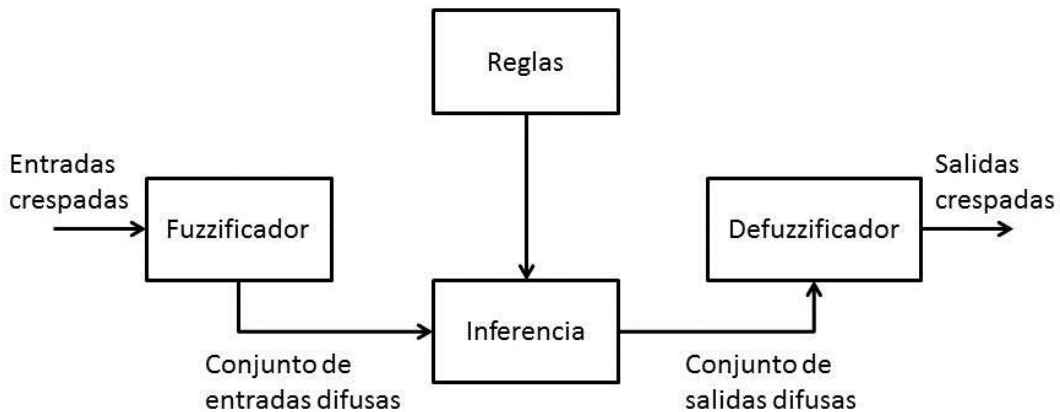


Diagrama T1FLS

Figura 5.12

- **Fuzzificador:** El fuzzificador mapea las entradas no difusas en los conjuntos difusos que activan el motor de inferencia.

- **Conjunto de reglas:** Las reglas en un T2FLS permanecen iguales que en el T1FLS, pero los antecedentes y los consecuentes son representados por un conjunto difuso de intervalo tipo 2.
- **Inferencia:** En los conjuntos difusos tipo-2, los operadores join (\sqcup) y meet (\sqcap) se usan en lugar de los operadores unión e intersección que son usados en el tipo 1.
- **Reducción de tipo:** Las salidas difusas tipo 2 del motor de inferencia son transformadas en conjuntos difusos tipo 1 que son llamados conjuntos de tipo reducido. Existen 2 métodos comunes para la operación de reducción de tipo en los intervalos T2FLS: El algoritmo de iteración de Karnik-Mendel, y el método de límites desconocidos de Wu-Mendel. Estos 2 métodos están basados en el cálculo del centroide.
- **Defuzzificación:** Las salidas del segmento de reducción de tipo son dadas al segmento de defuzzificación. Los conjuntos de tipo reducido son determinados por su punto de los extremos, izquierdo y derecho, el valor defuzzificado es calculado por el promedio de estos puntos.

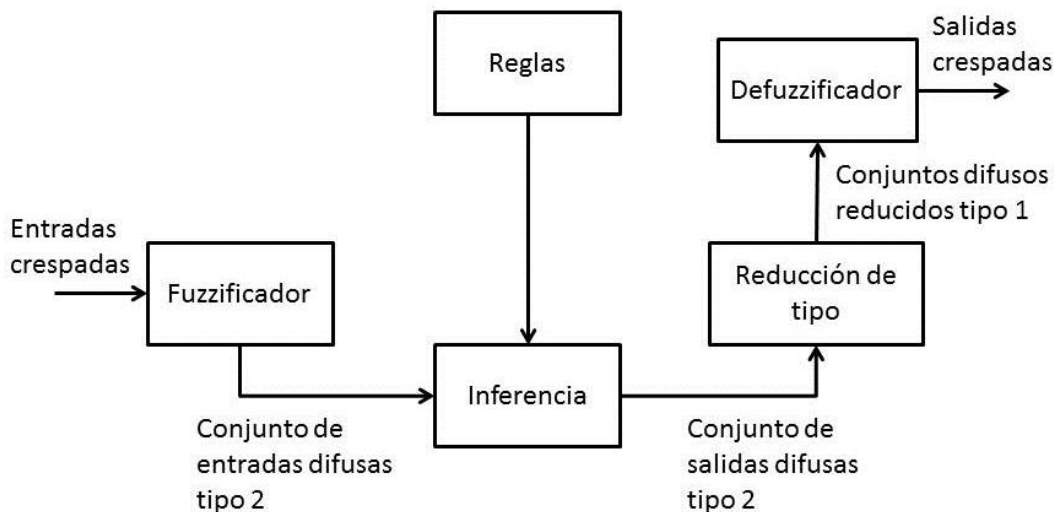


Diagrama T2FLS

Figura 5.13

5.5 Controladores difusos

Diseño de controladores difusos

En [20] se propone un algoritmo ACS que tiene un módulo con un controlador difuso para actualizar automáticamente los parámetros β y q_0 de acuerdo a algunos resultados de desempeño. Estos dos parámetros fueron elegidos debido a la fuerte relación observada entre ellos y el desempeño del algoritmo.

q_0 = Define un porcentaje de probabilidad en el cual se seleccionará al mejor elemento posible en la siguiente posición de la construcción

β = Es un valor que afecta a la probabilidad de emplear una medición heurística (distancia de un nodo a otro) en vez de usar la feromona para la selección del siguiente elemento.

Por ejemplo, considerando el parámetro q_0 , si la varianza de la población es alta esto significa que hay una variedad de diferentes caminos que actualmente están siendo explorados por las hormigas, por consecuencia no se necesita más exploración, se necesita explotar la información aprendida (los rastros de feromona y la información heurística), así se puede ajustar el parámetro q_0 a un valor más alto tal que se incremente la probabilidad de que la hormiga elija el mejor movimiento indicado por la feromona obtenida y la información heurística.

Por otra parte, con relación al parámetro β , si la varianza de la población es baja, significa que la hormiga puede estar estancada en un óptimo local causado por la feromona acumulada en ese camino, así que se necesita incrementar el valor del parámetro β tal que el peso de la información heurística (longitud del camino) es más alto que el peso del rastro de feromona en ese camino.

Para realizar el proceso de inducción de las reglas difusas fueron propuestos en [20] los siguientes pasos:

- Determinar las entradas y salidas de las reglas difusas: donde las entradas son el error y la varianza que son las medidas del desempeño, mientras que las salidas son los parámetros β y q_0 .
- Definir las funciones de membresía: se usan funciones de membresía triangular para cada entrada y salida.

- Preparar un archivo con múltiples ejemplos de la ejecución de un algoritmo ACS. Se crea el archivo variando cada parámetro a controlar de tal manera que el archivo muestre el desempeño del algoritmo con diferentes valores en los parámetros. Mientras más grande sea el archivo, mayor la probabilidad de producir reglas que describan mejor el efecto de los parámetros del algoritmo en el desempeño.

Inducción de reglas difusas

Para que un controlador difuso arroje los mejores valores de los parámetros para el algoritmo que se está ejecutando, en [20] se recomienda una regla base que describa el mejor desempeño del algoritmo de sistema de colonia de hormigas (ACS) en respuesta a los cambios en los parámetros β y q_0 . Para esto se necesitan valores difusos para estos parámetros; por ejemplo, podemos usar “*Bajo*”, “*Medio*”, “*Alto*”, “*No Bajo*”, “*No Medio*” y “*No Alto*”.

Los valores “*Bajo*”, “*Medio*” y “*Alto*” son guardados con los valores de “1”, “2” y “3” respectivamente. Los “*No*” pueden ser representados con valores negativos, por ejemplo, “*No Bajo*” se puede almacenar como “-1”.

Por ejemplo, si se quiere representar las reglas serían:

IF *Error* es “*Alto*” y *Varianza* es “*No Medio*” la salida sería:

3	-2
---	----

Para la regla:

IF *Varianza* es “*Baja*” THEN β es “*Baja*” y q_0 es “*Medio*”, la salida sería:

Error Varianza β q_0

0	1	1	2
---	---	---	---

Evaluación de una solución

Cuando se codifica una solución, debe indicarse cuantas reglas diferentes tiene asociada. El número de reglas asociadas a una solución es un número generado aleatoriamente entre 1 y el máximo número de reglas permitidas. Así cada solución tendrá un entero en el arreglo que represente el número de reglas en su conjunto de reglas y un arreglo de enteros que representa el conjunto entero de reglas. Se representaría de la siguiente manera:

$$LEN = (NOI + NOO) * (NOR) + 1$$

Donde *NOI* es el número de entradas, *NOO* el número de salidas, y *NOR* el número de reglas.

Dadas una solución y un ejemplo del archivo de muestras, si el ejemplo se puede producir con alguna de las reglas en el conjunto de reglas de la solución, el ejemplo se considera bien clasificado. En otro caso el ejemplo se considera mal clasificado por la solución. En general el valor objetivo de una solución está dado de la siguiente manera:

$$V.O. = (\text{número de ejemplos} - \text{número ejemplos mal clasificados}) / \text{número de ejemplos}$$

Un algoritmo que maximice este valor, determina la solución cuyo conjunto de reglas se adapta mejor al archivo de muestras utilizadas.

6. Controlador difuso para ACS-TSP, Constructivo Voráz-VB y Exhaustivo-VSP

Diseño e implementación de un controlador difuso

Se implementó un motor difuso tipo 1 para el ajuste de parámetros de los algoritmos: colonia de hormigas y 2 constructivo voraz. Para los problemas de TSP, bisección de vértices, y VSP. Dos de los algoritmos mencionados fueron tomados de la bibliografía, uno presentado por Dorigo[1], y el otro presentado por Terán. El algoritmo constructivo voraz fue implementado en este trabajo de investigación.

Se trabajó con el planteamiento del motor difuso para que no solo pudiera resolver el problema de TSP con ACS y VSP con Exhaustivo, sino que el motor difuso fuera capaz de trabajar con cualquier tipo de problema, dándole una configuración respecto al problema a resolver así como generando reglas para dicho algoritmo.

A continuación se mostrará la implementación del motor difuso para:

- El problema de TSP con ACS, el cual muestra el ajuste de parámetros global del motor difuso al final de cada ejecución de la instancia evaluada.
- El problema de Bisección de Vértices con Tabú, el cual muestra el ajuste de parámetros local al final de cada iteración del problema. Donde la evaluación del cálculo de los parámetros está incluida en la heurística tabú.
- El problema VSP con un constructivo voraz, donde la evaluación del cálculo se encuentra después de cada iteración de la construcción.

El ajuste de parámetros del controlador difuso es en línea en todos los 3 problemas mencionados; es decir, el ajuste de parámetros sucede sobre la ejecución del algoritmo.

El controlador difuso cuenta con un ajuste para las funciones de membresía, el cual permite indicar el centro y la desviación estándar en las funciones gaussianas, el centro y los 2 puntos inferiores izquierdo y derecho de las funciones triangulares, los 2 puntos inferiores izquierdo y derecho, y los 2 puntos superiores izquierdo y derecho en las funciones trapezoidales.

Dentro de la adaptabilidad del motor difuso existe la configuración de los operadores difusos AND, OR y NOT en el conjunto de reglas difusas; las cuales afectan la evaluación de los parámetros, se muestran algunos ejemplos a continuación:

<i>Operador</i>	<i>Error</i>	<i>Varianza</i>	β	q_0
<i>AND</i>	1	2	1	0
<i>OR</i>	1	2	2	0
<i>NOT</i>	-3	-1	-3	0

Tabla 6.1

La configuración del parámetro NOT no es necesaria especificarla en el archivo de lectura de reglas, dado que con poner en negativo el valor lo identifica como NOT.

Se realizó una experimentación previa para encontrar un conjunto de 9 reglas que arrojaron los mejores resultados para los distintos algoritmos, esto no significa que no exista un conjunto de reglas que produzca mejores resultados.

β	<i>Error</i>	$\beta\Delta$
1	1	3
1	2	2
1	3	3
2	1	1
2	2	1
2	3	3
3	1	2
3	2	3
3	3	2

Tabla 6.2

Los problemas de Vertex Bisection con Tabú y VSP con Constructivo Voráz usaron el conjunto de 9 reglas especificado y cuentan con los resultados generados por este motor. A su vez, los resultados obtenidos fueron verificados con el motor difuso de Matlab.

6.1 TSP con ACS.

Se tomó lo antes ya mencionado por Amir et al en [5] para elegir los valores discretos de error, varianza, β y q_0 generados por el ACS para el problema de TSP.

<i>Error</i>	<i>Varianza</i>	β	q_0
0.008524164	13084.667	2	0.8

Tabla 6.1.1

Los 4 parámetros que se observan en la tabla anterior se utilizaron en la etapa de fuzzificación con la finalidad de generar las funciones de membresía “low”, “mid” y “hi” para cada parámetro; las funciones de membresía se representaron con los valores numéricos 1, 2 y 3.

<i>Error</i>	<i>Varianza</i>	β	q_0
1	1	1	3

Tabla 6.1.2

El siguiente paso fue la creación de las funciones de membresía Gaussianas para los 4 parámetros, a las cuales se les otorgó una desviación estándar fija dependiendo del rango en el cual estén configuradas; se normalizó el error y para la varianza se generó una desviación estándar adaptable según sea el rango obtenido. Los valores de la desviación estándar fueron elegidos para que las funciones de membresía estén simétricas y con traslape.

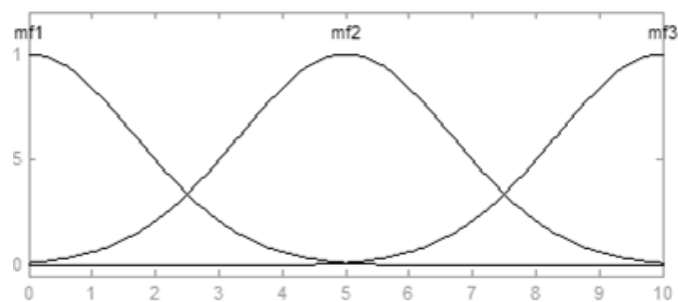


Figura 6.1.1

Se implementó un conjunto de reglas los cuales reflejasen mejores resultados al conjunto de reglas generadas a partir del planteamiento del manejo de la β y q_0 por Amir et al. Es importante recalcar que todas estas reglas emplean el operador AND y por

consecuente las operaciones mostradas posteriormente se llevarán a cabo bajo esta consideración.

<i>Error</i>	<i>Varianza</i>	β	q_0
1	2	1	0
3	3	2	0
2	3	1	0
1	1	1	0

Tabla 6.1.3

Se genera una tabla de topes donde por cada regla del conjunto de reglas se evalúan los valores de los parámetros correspondientes a *error* y *varianza* según la clasificación difusa de la regla (*low*, *mid* y *hi*) y se elige el menor de los valores resultantes si las reglas empleadas son AND, o se elige el mayor de los valores resultantes si las reglas empleadas son OR.

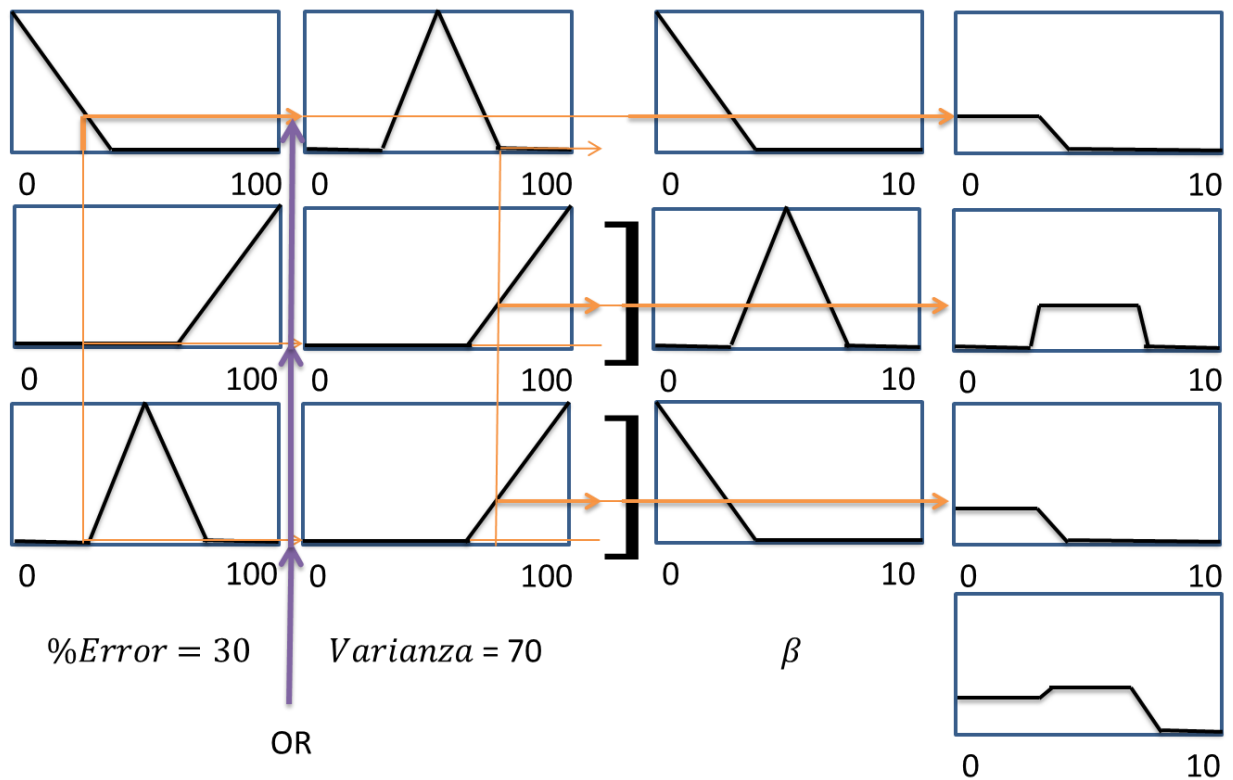


Figura 6.1.2

A continuación se generan 2 arreglos para β y q_0 . Esto con el objetivo de generar los valores de x y y que representarán sus gráficas resultantes. Se evalúa la función de membresía para cada punto de los arreglos, con un incremento de β y de q_0 según su clasificación en la regla actual.

Error	Varianza	β	q_0
2	3	1	0

Tabla 6.1.4

β	1.0
Evaluación en 1	0.807451

Tabla 6.1.5

Se toma el menor valor entre el valor obtenido y el valor Tope Resultado de la tabla de topes, y después se toma al mayor entre el valor elegido y el que se encuentra en el arreglo de β o q_0 según sea el caso y se actualiza el valor del arreglo.

β	1.009009	1.01	...
Evaluación en 1	0.807451	0.803987	...

Tabla 6.1.6

A continuación se muestra la representación gráfica de los valores encontrados en el arreglo de β con los valores de 1, 5 y 10:

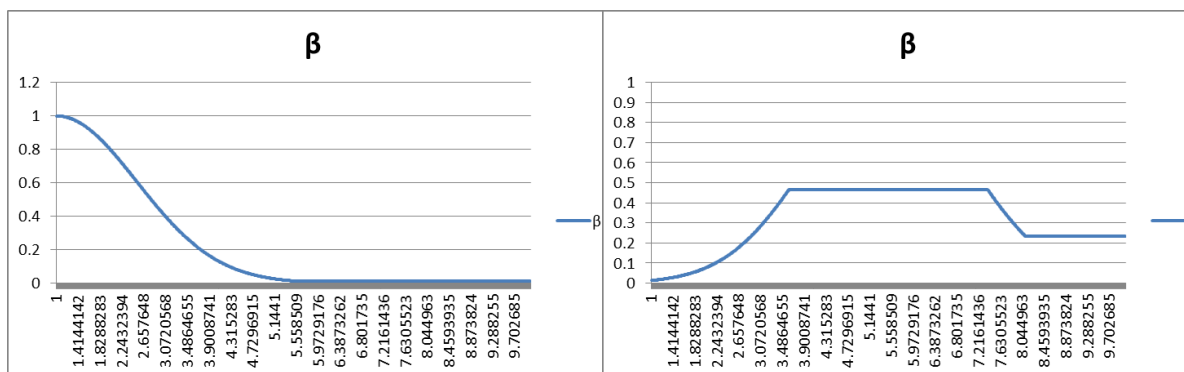


Figura 6.1.3

Figura 6.1.4

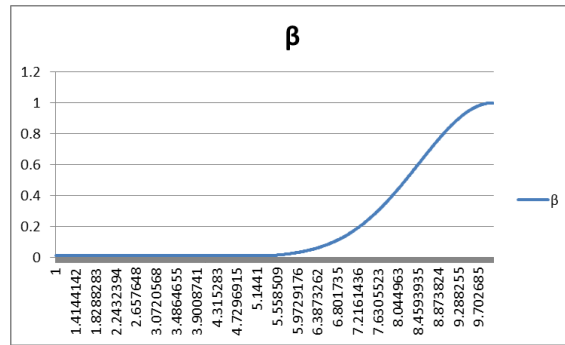


Figura 6.1.5

Como podemos observar, β muestra una función de pertenencia correspondiente al comportamiento de las reglas respecto a los valores ingresados (1 (Figura 6.1.3), 5 (Figura 6.1.4) y 10 (Figura 6.1.5)).

Posteriormente el controlador difuso necesita la función de centroide para la defuzzificación, la cual elige el punto medio de todos los valores obtenidos en los arreglos de β y q_0 , para actualizar los mismos valores para la siguiente iteración.

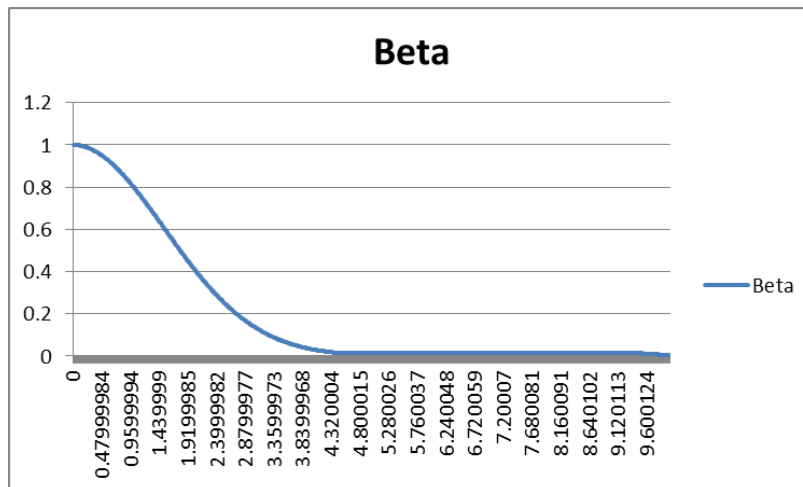


Figura 6.1.6

En la figura 6.1.6 podemos observar que la función de membresía de Beta pertenece a "low", una vez defuzzificada la gráfica por medio del centroide obtenemos el valor de 1.4577441. El valor de β es actualizado en la ejecución siguiente por el valor obtenido por la defuzzificación.

A continuación se muestran los resultados de 10 instancias TSP con el algoritmo original de ACS con los parámetros de $\beta = 5$ y $q_0 = 0.9$; se eligió el mejor resultado obtenido en 5 ejecuciones de cada una.

<i>Instancia</i>	<i>Óptimo conocido</i>	<i>Valor Obtenido</i>	<i>% error</i>
att532	27686	27733	0.169
d198	15780	15780	0
d1291	50801	51097	0.582
eil51	426	426	0
kroA100	21282	21282	0
lin318	42029	42029	0
pcb442	50778	50838	0.118
pcb1173	56892	57639	1.313
pr2392	378032	386930	2.353
rat783	8806	10393	18.021

Tabla 6.1.7

La siguiente tabla muestra los resultados de las instancias con el motor difuso tipo 1 con óptimos conocidos, el parámetro de q_0 se mantiene fijo en 0.9.

<i>Instancia</i>	<i>Óptimo conocido</i>	<i>Valor Obtenido</i>	<i>% error</i>
att532	27686	27690	0.01
d198	15780	15780	0
d1291	50801	50958	0.3
eil51	426	426	0
kroA100	21282	21282	0
lin318	42029	42029	0
pcb442	50778	50838	0
pcb1173	56892	57534	1.12
pr2392	378032	388897	2.8
rat783	8806	10381	17.8

Tabla 6.1.8

La siguiente tabla muestra los resultados de las instancias con el motor difuso tipo 1 con el mejor valor obtenido, el parámetro de q_0 se mantiene fijo en 0.9.

<i>Instancia</i>	<i>Óptimo conocido</i>	<i>Valor Obtenido</i>	<i>% error</i>
att532	27686	27712	0.09
d198	15780	15780	0
d1291	50801	50964	0.32
eil51	426	426	0
kroA100	21282	21282	0
lin318	42029	42029	0
pcb442	50778	50785	0.01
pcb1173	56892	57632	1.3
pr2392	378032	387340	2.4
rat783	8806	10370	17

Tabla 6.1.9

Podemos observar una mejoría en los casos de óptimos conocidos como los no conocidos usando el motor difuso contra los resultados sin el motor difuso. A continuación se muestra la tabla de promedios para este tipo de instancias.

Tabla 6.1.10

<i>Instancias</i>	<i>Promedio sin motor difuso</i>	<i>Promedio con motor difuso</i>
<i>TSP</i>	66602.3	66300.2

Se realizaron pruebas estadísticas de Wilcoxon para evaluar el desempeño estadístico del motor difuso. Y se determinó que si existe una diferencia significativa del desempeño del algoritmo con el sistema difuso en comparación con el algoritmo sin el sistema difuso, con una certidumbre superior al 95%.

6.2 Vertex Bisection con Algoritmo Constructivo Voráz.

Para demostrar que el motor difuso es adaptable a cualquier problema, se usó el algoritmo voraz para el problema Vertex Bisection generado por Terán, que es un algoritmo de búsqueda GRASP con una heurística voraz de construcción inicial.

A continuación se muestra un algoritmo constructivo voraz diseñado para el problema de Bisección de Vértices (ver Figura 3).

```

Elem = 0
AdyToRv = Gradov       $\forall v \in V$ 
R ← {v}                 $\forall v \in V$ 
L = ∅
u = ArgMinv(Gradov)
L ← {u}
R = R \ {u}
AdyToRv--               $\forall v \in V | \exists (u, v) \in E$ 
Elem++
do{
  LC = R
  Limite = Min(AdyToRv) + β (Max(AdyToRv) - Min(AdyToRv))
                 $\forall v \in LC$ 
  LCR = LC \ v ∈ LC | AdyToRv > Limite
  u = SelectAleatoria(LCR)
  L ← {u}
  R = R \ {u}
  AdyToRv--               $\forall v \in V | \exists (u, v) \in E$ 
  Elem++
}while(Elem < [|V|/2])

```

Algoritmo 6.2.1: Algoritmo constructivo Voraz.

Al inicio del algoritmo se inicializa la cantidad de elementos seleccionados (*Elem*), todos los vértices se encuentran en el conjunto derecho y se considera que cada vértice tiene a todos sus adyacentes (*Grado_v*) en el lado derecho (*AdyToR*). El primer elemento seleccionado será aquél con menor grado, aquél que tenga menos enlaces a elementos de la derecha, este elemento será removido del conjunto derecho y asignado al izquierdo.

Además, se realizará una actualización de todos los adyacentes del elemento seleccionado, esto se hace con el fin de identificar aquéllos vértices que tienen menos

conexiones hacia el conjunto de la derecha (*AdyToR*). Después se creará la lista de candidatos (*LC*) con todos los elementos que todavía se encuentran en el conjunto de la derecha (*R*). Y la lista de candidatos restringida (*LCR*) contendrá los mismos elementos que *LC*, menos aquellos elementos que tengan más adyacentes a la derecha que los definidos por la variable Límite. Siendo el Límite un valor entre el mínimo y máximo número de adyacentes a la derecha de todos los vértices en *LC*, el valor del límite depende de la variable β que es un valor entre cero y uno. Si la variable β tiende a cero entonces el límite será muy restrictivo, en cambio, si β tiende a uno entonces el límite será muy laxo. Luego se selecciona un vértice (*u*) de *LCR* de manera aleatoria, se elimina del conjunto de la derecha (*R*) y se incluye en el conjunto de la izquierda (*L*). Posteriormente se actualizan los adyacentes hacia la derecha *AdyToR* disminuyendo en uno a todos los vértices que son adyacentes a *u*. Se continúa con este proceso hasta que se hayan movido, al conjunto de la izquierda, la mitad de vértices del grafo.

Al algoritmo anterior se le incorporó un controlador difuso para ajustar automáticamente el parámetro β . El controlador difuso recibe como variable de entrada no difusa la β y el $\%Err$ de la solución actual con respecto a la mejor observada en el proceso de búsqueda y como variable de salida no difusa $\Delta\beta$, el cual especifica un incremento en la β . En cada iteración el algoritmo genera la solución actual, se actualiza la mejor solución encontrada y se determina el porcentaje de error ($\%Err$).

El controlador usa 3 funciones de membresía gaussianas (bajo, medio, alto) tanto para el parámetro de entrada como el de salida. Estas funciones de membresía tienen la característica de ser simétricas y con traslape. Las siguientes figuras (Figura 4, Figura 5 y Figura 6) muestran las funciones de membresía utilizadas.

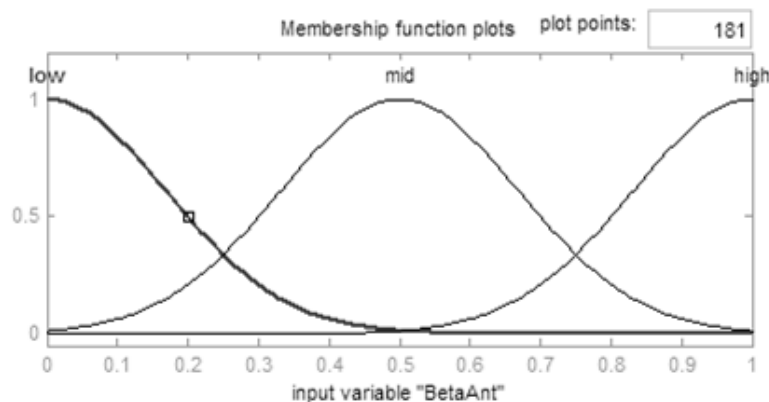


Figura 6.2.1: Variable de entrada " β "

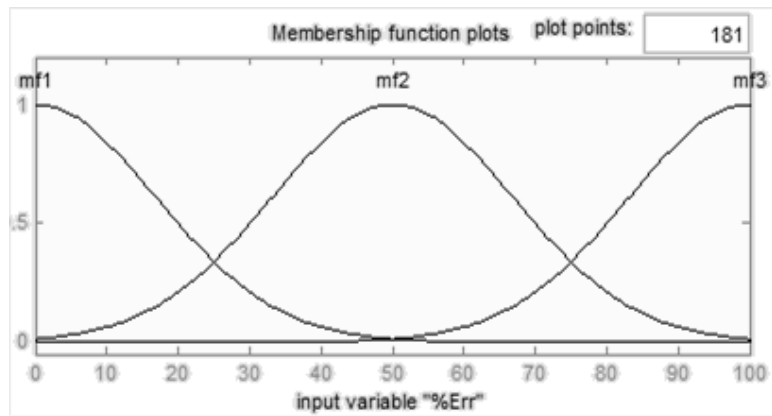


Figura 6.2.2: Variable de entrada “%Err”

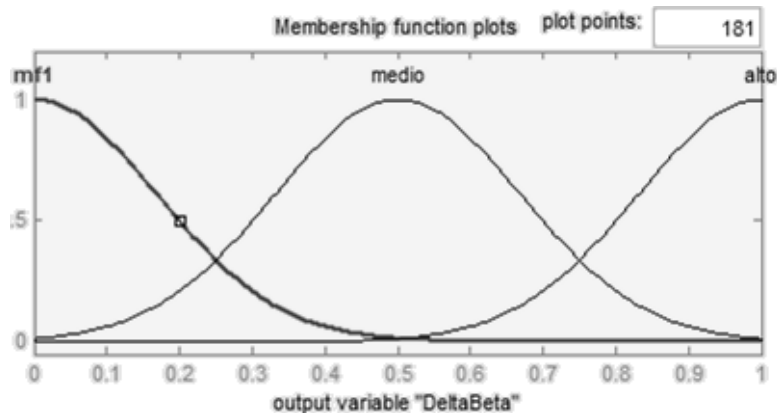


Figura 6.2.3: Variable de salida “ $\Delta\beta$ ”

Las reglas que utiliza el controlador son las siguientes:

- Sí, β es bajo y %Err es bajo entonces $\Delta\beta$ será alto;
- Sí, β es bajo y %Err es medio entonces $\Delta\beta$ será medio;
- Sí, β es bajo y %Err es alto entonces $\Delta\beta$ será alto;
- Sí, β es medio y %Err es bajo entonces $\Delta\beta$ será bajo;
- Sí, β es medio y %Err es medio entonces $\Delta\beta$ será bajo;
- Sí, β es medio y %Err es alto entonces $\Delta\beta$ será alto;
- Sí, β es alto y %Err es bajo entonces $\Delta\beta$ será medio;
- Sí, β es alto y %Err es medio entonces $\Delta\beta$ será alto;
- Sí, β es alto y %Err es alto entonces $\Delta\beta$ será medio;

En este algoritmo se contará con 3 variables en el conjunto de reglas: 2 de entrada (β , *Porcentaje de Error*) y una de salida ($\Delta\beta$).

β	<i>%Error</i>	$\Delta\beta$
0.3	88.0	0.5867

Tabla 6.2.1

Dado que el cálculo de la β y el *porcentaje de error* se genera en el algoritmo constructivo voraz, se procede desde la creación de las funciones de membresía; esto al mismo tiempo muestra la versatilidad del motor difuso dado a que puede o no evaluar las variables necesarias para el cálculo de las variables de salida.

Como las variables de entrada y salida existen en el rango de 0 a 1 y 0 a 100, las funciones gaussianas estarán configuradas con centro de 0, 0.5 y 1 con una desviación estándar de 1.699, y 0, 50 y 100 con una desviación estándar de 16.99 respectivamente, para que estén simétricas y con traslape.

Se genera una tabla de topes donde por cada regla del conjunto de reglas se evalúan los valores de los parámetros correspondientes a β y *Porcentaje de Error* según la clasificación difusa de la regla (*low*, *mid* y *hi*) y se elige el menor de los valores resultantes dado a que las reglas usan operadores AND.

<i>Reglas</i>	<i>Tope β</i>	<i>Tope %Error</i>	<i>Tope Resultado</i>
1	0.003915	0.006721	0.003915
2	0.458652	0.006721	0.006721
⋮	⋮	⋮	⋮

Tabla 6.2.2

A continuación se genera 1 arreglo de 2 x 1000 para $\Delta\beta$. Esto con el objetivo de generar los valores de x y y que representarán su gráfica resultante. Se evalúa la función de membresía para cada punto de los arreglos, con un incremento de 0.001 para $\Delta\beta$ por su clasificación en la regla actual dado que sus rangos deben estar entre 0 y 1.

β	%Error	$\Delta\beta$
2	3	3

Tabla 6.2.3

$\Delta\beta$	0.9
	0.782532

Tabla 6.2.4

Se toma al menor entre el valor obtenido y el valor Tope Resultado de la tabla de topes, y después se toma al mayor entre el valor elegido y el que se encuentra en el arreglo de *percentTTabu*, actualizando el valor del arreglo.

$\Delta\beta$	0.9	0.901	...
	0.782532	0.782546	...

Tabla 6.2.5

Posteriormente el controlador difuso necesita la función de centroide para la defuzzificación, la cual elige el punto medio de todos los valores obtenidos en el arreglo de $\Delta\beta$ para actualizar los mismos valores para la siguiente iteración.

El valor de $\Delta\beta$ es actualizado en la ejecución siguiente por el valor obtenido por la defuzzificación.

A continuación se muestran los resultados utilizando este conjunto de reglas en el problema para este motor difuso y el motor de Matlab:

<i>Instancias</i>	% sin <i>Motor Difuso</i>	% <i>Motor Difuso</i>	% <i>Matlab</i>
<i>Grid</i>	11	8.64	8.6
<i>Harwell</i> – <i>Boeing</i>	149.6551724	136.88505	136.597701
<i>Small</i>	5.30952381	4.3364872	4.30952381

Tabla 6.2.6

En estos resultados se observa que en casi todos los casos se reduce o se empata el valor objetivo al emplear un controlador difuso, ya sea en Java o Matlab. También podemos ver

que se reduce el valor promedio de la función objetivo al emplear cualquier variante de los controladores difusos.

Los resultados de la prueba no paramétrica de Wilcoxon muestran que las mejoras observadas al utilizar los controladores difusos son estadísticamente significativas (con una certidumbre superior al 99%). Por otra parte, la prueba de Wilcoxon muestra resultados de equivalencia estadística entre los resultados obtenidos con la aplicación en Java y los resultados de Matlab. Estos resultados muestran que la implementación en Java posee la misma calidad que Matlab, y por lo tanto es una alternativa viable para realizar pruebas de sistemas difusos.

6.3 VSP con Constructivo Voraz.

El motor difuso se configuró para poder trabajar con el problema VSP con la heurística Constructiva Voraz, tomando β y Porcentaje de Error como valor de entrada y $\Delta\beta$ como valor de salida. La heurística Constructiva Voraz para el problema VSP fue implementada en Java y usa el enfoque de evaluación local por iteración terminada.

El algoritmo de la heurística Constructiva Voraz es el siguiente:

```
Do{
  Inicializar_Permutacion()
  Calcular_Limite_Lista_Adyacencia()
  Seleccionar_Primer_Elemento()
  for(i = 1 to n - 1){
    Calcular_Grado_Permutacion()
    Calculo_Siguiente_Elemento()
    Calcular_Limite()
    Seleccionar_Siguiente_Elemento()
    Posicionar_Elemento()
  }
  Calcular_VSP()
  If(max_iter > 1){
    Porcentaje_error()
    Motor_Difuso()
    Actualizar_Beta()
  }
  max_iter++
} while (max_iter < condición_paro)
Elegir_Menor_VSP()
```

Algoritmo 6.3.1

Dado que el cálculo de la β y el *porcentaje de error* se genera en el algoritmo constructivo voraz, se procede desde la creación de las funciones de membresía; esto al mismo tiempo muestra la versatilidad del motor difuso dado que puede o no evaluar las variables necesarias para el cálculo de las variables de salida.

Como las variables de entrada y salida existen en el rango de 0 a 1 y 0 a 100, las funciones gaussianas estarán configuradas con centro de 0, 0.5 y 1 con una desviación estándar de 1.699, y 0, 50 y 100 con una desviación estándar de 16.99 respectivamente, para que estén simétricas y con traslape. Se implementó un conjunto de reglas los cuales reflejasen mejores resultados que el algoritmo sin el controlador difuso.

β	% Error	$\Delta\beta$
1	1	3
1	2	2
1	3	3
2	1	1
2	2	1
2	3	3
3	1	2
3	2	3
3	3	2

Tabla 6.3.1

Es decir, las reglas que utiliza el controlador son las siguientes:

1. Sí, β es bajo y %Err es bajo entonces $\Delta\beta$ será alto;
2. Sí, β es bajo y %Err es medio entonces $\Delta\beta$ será medio;
3. Sí, β es bajo y %Err es alto entonces $\Delta\beta$ será alto;
4. Sí, β es medio y %Err es bajo entonces $\Delta\beta$ será bajo;
5. Sí, β es medio y %Err es medio entonces $\Delta\beta$ será bajo;
6. Sí, β es medio y %Err es alto entonces $\Delta\beta$ será alto;
7. Sí, β es alto y %Err es bajo entonces $\Delta\beta$ será medio;
8. Sí, β es alto y %Err es medio entonces $\Delta\beta$ será alto;
9. Sí, β es alto y %Err es alto entonces $\Delta\beta$ será medio;

Se genera una tabla de topes donde por cada regla del conjunto de reglas se evalúan los valores de los parámetros correspondientes a β y *Porcentaje de Error* según la clasificación difusa de la regla (*low*, *mid* y *hi*) y se elige el menor de los valores resultantes.

Reglas	Tope β	Tope %Error	Tope Resultado
1	0.003915	0.006721	0.003915
2	0.458652	0.006721	0.006721
⋮	⋮	⋮	⋮

Tabla 6.3.2

A continuación se genera 1 arreglo de 2 x 1000 para $\Delta\beta$. Esto con el objetivo de generar los valores de x y y que representarán su gráfica resultante. Se evalúa la función de

membresía para cada punto de los arreglos, con un incremento de 0.001 para $\Delta\beta$ por su clasificación en la regla actual dado que sus rangos deben estar entre 0 y 1 .

β	%Error	$\Delta\beta$
2	3	3

Tabla 6.3.3

$\Delta\beta$	0.9
	0.782532

Tabla 6.3.4

Se toma al menor entre el valor obtenido y el valor Tope Resultado de la tabla de topes, y después se toma al mayor entre el valor elegido y el que se encuentra en el arreglo de *percentTTabu*, actualizando el valor del arreglo.

$\Delta\beta$	0.9	0.901	...
	0.782532	0.782546	...

Tabla 6.3.5

Posteriormente el controlador difuso necesita la función de centroide para la defuzzificación, la cual elige el punto medio de todos los valores obtenidos en el arreglo de $\Delta\beta$ para actualizar los mismos valores para la siguiente iteración.

El valor de $\Delta\beta$ es actualizado en la ejecución siguiente por el valor obtenido por la defuzzificación.

Para la evaluación con el sistema difuso de Matlab se genera un archivo .fis el cual contendrá el sistema difuso incluyendo las entradas del controlador que son β y %Err y la salida que es $\Delta\beta$.

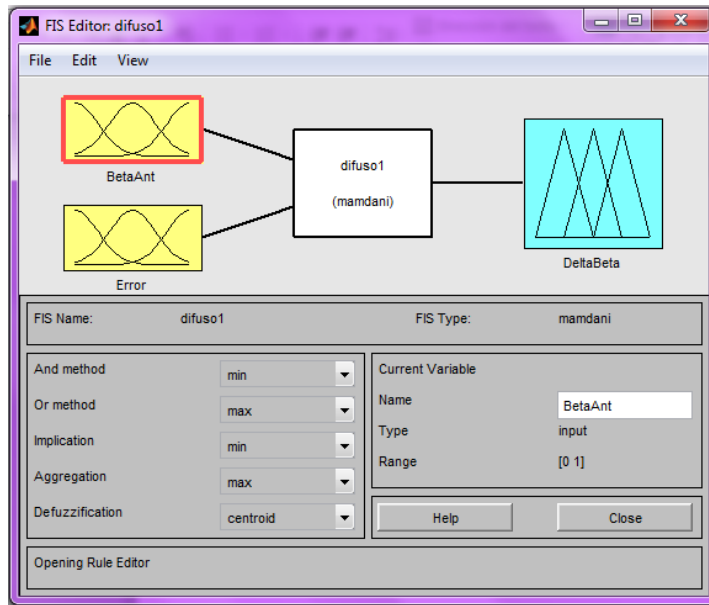


Figura 6.3.1

El controlador usa 3 funciones de membresía gaussianas (bajo, medio, alto) tanto para el parámetro de entrada como el de salida. Estas funciones de membresía tienen la característica de ser simétricas y con traslape. Las siguientes figuras (Figura 4, Figura 5 y Figura 6) muestran las funciones de membresía utilizadas.

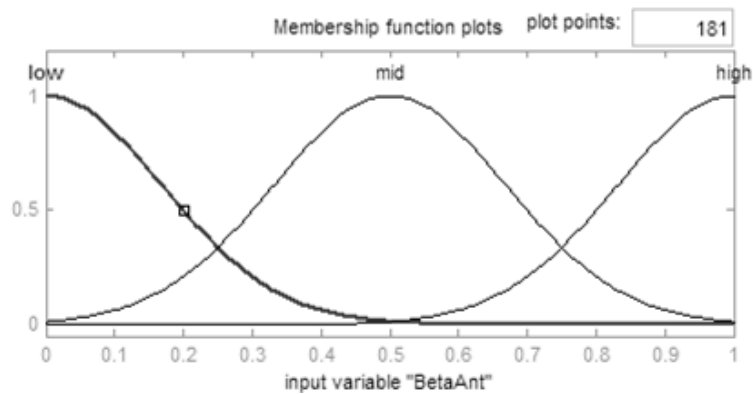


Figura 6.3.2: Variable de entrada " β "

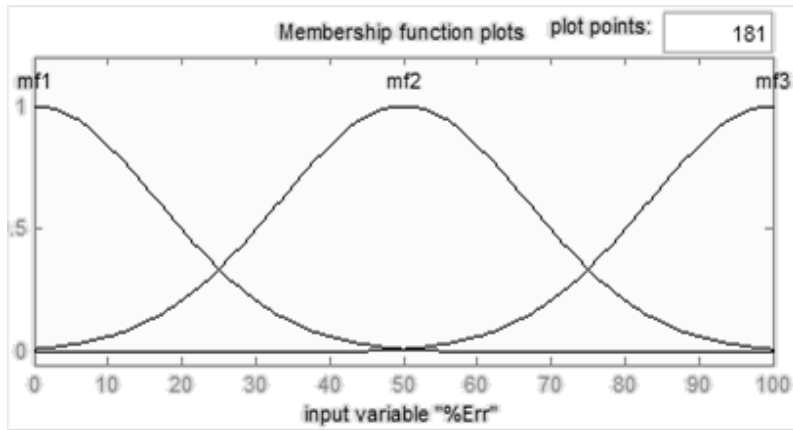


Figura 6.3.3: Variable de entrada “%Err”

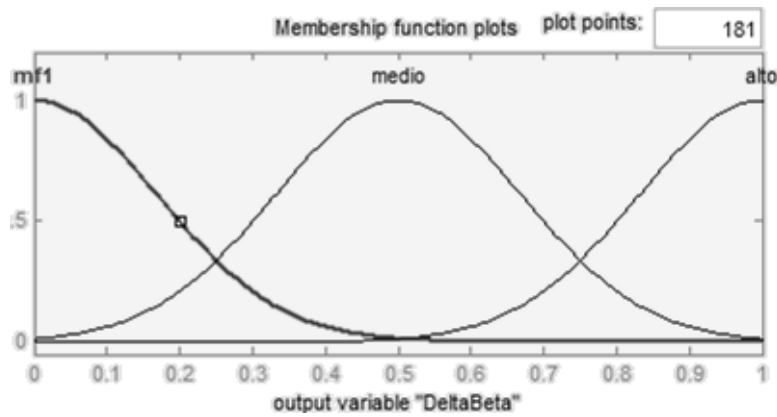


Figura 6.3.4: Variable de salida “ $\Delta\beta$ ”

La tabla siguiente muestra los porcentajes para los conjuntos de instancias descritos con y sin el motor difuso, y la evaluación con el sistema difuso de Matlab:

<i>Instancias</i>	<i>% sin Motor Difuso</i>	<i>% Motor Difuso</i>	<i>% Matlab</i>
<i>Grid</i>	21.74074074	22.9753086	22.5555556
<i>Harwell</i> <i>– Boeing</i>	451.7701149	417.942529	417.724138
<i>Small</i>	5.511904762	5.88095238	5.47619048

Tabla 6.3.6

Haciendo las pruebas con Wilcoxon, en las instancias *Grid* si existen diferencias significativas. Es estadísticamente mejor el desempeño del algoritmo sin el motor difuso, con un nivel de certidumbre superior al 99%. Con la experimentación realizada se observó que como son instancias muy fáciles, no realizan muchas iteraciones y por lo tanto es más

conveniente emplear solamente el mejor parámetro conocido a estar cambiando los parámetros y perder tiempo computacional revisando soluciones que incluyen "diversidad innecesaria" para estas instancias pequeñas.

En las instancias *Harwell – Boeing* existen diferencias significativas. El desempeño del algoritmo con motor difuso es estadísticamente mejor que sin el motor difuso, con un nivel de certidumbre superior al 99%. Se considera que este es el conjunto de instancias de mayor importancia debido a su gran tamaño, y aquí es donde se muestra la utilidad de emplear un motor difuso que controle los parámetros del algoritmo.

En las instancias *Small* no existen diferencias significativas, es estadísticamente equivalente emplear un sistema difuso o no para este conjunto de instancias.

7. Conclusiones y Trabajos Futuros

En esta sección se presentan las conclusiones de este trabajo, así como sugerencias para el desarrollo de los trabajos futuros en esta área.

7.1 Conclusiones

En este trabajo se desarrolló un sistema difuso para controlar los parámetros de una solución metaheurística del problema TSP, VB y VSP, cumpliendo con los objetivos planteados inicialmente, en donde se proponía lo siguiente:

- Diseñar e implementar un sistema difuso para controlar los parámetros de una solución metaheurística del problema VSP.
- Revisar el marco teórico de los temas relacionados con el proyecto.
- Revisar el estado del arte de controladores difusos de parámetros de algoritmos metaheurísticos aplicados a problemas relacionados con el VSP.
- Implementación de un sistema difuso para controlar los parámetros de un sistema de colonia de hormigas [16] aplicado a la solución de TSP.
- Diseño e implementación de un sistema difuso para controlar los parámetros de una metaheurística aplicada a la solución del problema VSP.
- Análisis experimental del impacto del controlador difuso en el desempeño del algoritmo correspondiente.

Además de esto, se implementó un motor difuso *adaptable* a cualquier problema con cualquier heurística, siempre y cuando sea configurado adecuadamente.

Se demostró estadísticamente que los algoritmos con el motor difuso tipo 1 arrojaron valores más cercanos al óptimo que sin el uso del motor difuso.

Así mismo el motor difuso fue diseñado para poder ejecutar tipo 1, intervalo tipo 2 y tipo 2; completándose el tipo 1, y llevando hasta el momento 85% del intervalo tipo 2; cabe recalcar que para esta tesis solo fue planteado como objetivo el tipo 1 para el problema TSP y VSP, así como las operaciones AND solamente. Sin embargo, a este motor se le ha incluido también los operadores OR y NOT, así como la intercomunicación por sockets para trabajar con algoritmos que no estén implementados en Java.

7.2 Trabajos Futuros

Para dar continuidad a este trabajo de investigación, a partir de la experiencia obtenida, se propone:

-Terminar la implementación del controlador difuso de intervalo tipo 2 y el controlador difuso tipo 2 para los problemas abordados en esta tesis.

-Diseñar controladores difusos tipo 1, intervalo tipo 2 y tipo 2 para los problemas abordados pero con la característica de que sean auto configurables en cuanto a número de reglas, tipos de reglas y funciones de membresía.

Anexo: Productos Científicos

Referencias bibliográficas

- [1] Richard J. Lipton, Robert E. Tarjan.: "A Separator Theorem for Planar Graphs". Computer Science Department. School of Humanities and Sciences. Stanford University. (1977).
- [2] Josep Díaz., Jordi Petit., María Serna.: "A Survey of Graph Layout Problems". Universitat Politècnica de Catalunya. ACM Computing Surveys, Vol 34, No. 3, pp. 313-356. (2002).
- [3] Abraham Duarte, Laureano F. Escudero, Rafael Martí, Nenad Mladenovic, Juan José Pantrigo, Jesús Sánchez-Oro.: "Variable Neighborhood Search for the Vertex Separation Problem". Computers and Operations Research. Elsevier. (2012).
- [4] Nilesh N. Karnik, Jerry M. Mendel.: "Centroid of a type-2 fuzzy set". Department of Electrical Engineering-Systems, Signal and Image Processing Institute, University of Southern California (2001):
- [5] S. Areibi, G. Grewal, D. Banerji, P. Du.: "Hierarchical FPGA Placement". (2006).
- [6] Sang-Joon Lee., Raahemifar K.: "FPGA placement optimization methodology survey". Proceedings of Canadian Conference on Electrical and Computer Engineering. (2008).
- [7] Cadence Design Systems, Inc.: "Cadence OrCAD FPGA System Planner". 22239 06/11 MK/DM/PDF. (2011).
- [8] Hans L. Bodlaender.: "Linear-Time register allocation for a fixed number of registers". Proceedings of the Symposium on Discrete Algorithms. (1998).
- [9] András Kornai.: "Narrowness, Path-width, and their Application in Natural Language Processing". Discrete Applied Mathematics 36, pp. 87-92. Elsevier Science Publishers B. V. (1992).
- [10] Duarte Muñoz, A., Pantrigo Fernández, J., and Gallego Carrillo, M. Metaheurísticas. 2007.
- [11] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, 35(3):268–308, 2003.
- [12] Garey, M. R., Johnson, D. S., et al. Computers and Intractability: A Guide to the Theory of NP-completeness, 1979.
- [13] M. Laguna, C. Delgado. Introducción a los metaheurísticos. Monográfico 3. 2007.
- [14] James P. Ignizio and Tom M. Cavalier, Linear Programming, Edit. Prentice Hall, 1994.
- [15] H. Zanakis, J.R. Stelios, and A. Evans. Heuristic optimization: Why, when and how to use it. Interfaces, 5(11):84–90, 1981.

- [16] J.P. Kelly and I.H. Osman. *Meta-Heuristic: Theory and Applications*. Kluwer Academic Publisher, 1996.
- [17] Marco Dorigo and Luca Maria Gambardela: *Ant Colony System: "A Cooperative Learning Approach to the Traveling Salesman Problem"*. Université Libre de Bruxelles, Belgium, 1996.
- [18] Zbigniew Michalewicz and David B. Fogel: *"How to Solve It: Modern Heuristics"*. Springer, September, 1999.
- [19] R. Rojas: *Neural Networks*, Springer-Verlag, Berlin, 1996
- [20] Erdal Kayacan: *Interval Type-2 Fuzzy Logic Systems: Theory and Design*. Boğaziçi University, 2011.
- [21] Cherry Amir, Amr Badr and Ibrahim Farag: *A Fuzzy Logic Controller for Ant Algorithms*. Faculty of Computers and Information, Department of Computer Science, Cairo University.
- [22] Norberto Castillo García: *Optimización del Problema de la Separación de los Vértices de un Grafo Conexo no Dirigido*. Instituto Tecnológico de Ciudad Madero, 2012.
- [23] L. A. Zadeh: *Fuzzy Sets*. University of California, Berkeley, California. 1965.
- [24] Oscar Castillo and Patricia Melin: *Recent Advances in Interval Type-2 Fuzzy Systems*. Springer. Tijuana Institute of Technology, 2012.
- [25] C.-F. Juang, C.-H. Hsu, C.-F. Chuang, Reinforcement self-organizing interval type-2 fuzzy system with ant colony optimization, in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, 2009, pp. 771–776
- [26] R. Martinez-Marroquin, O. Castillo, J. Soria, Parameter tuning of membership functions of a type-1 and type-2 fuzzy logic controller for an autonomous wheeled mobile robot using ant colony optimization, in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, 2009, pp. 4770–4775
- [27] C.-F. Juang, C.-H. Hsu, Reinforcement ant optimized fuzzy controller for mobile-robot wallfollowing control. *IEEE Trans. Ind. Electron.* 56(10), 3931–3940 (2009)
- [28] O. Castillo, R. Martinez-Marroquin, P. Melin, F. Valdez, J. Soria, Comparative study of bioinspired algorithms applied to the optimization of type-1 and type-2 fuzzy controllers for an autonomous mobile robot. *Inf. Sci.* 192(1), 19–38 (2012)
- [29] C.-F. Juang, C.-H. Hsu, Reinforcement interval type-2 fuzzy controller design by online rule generation and Q-value-aided ant colony optimization. *IEEE Trans. Syst. Man Cybern. B Cybern.* 39(6), 1528–1542 (2009)