ALGORITMO DE OPTIMIZACIÓN GOTAS DE AGUA VIRTUALES INTELIGENTES APLICADO A LA PLANEACIÓN DE RUTA ÓPTIMA DE UN ROBOT MÓVIL

JUAN HERMÓGENES ARIAS BARAJAS

MOISÉS FRANCISCO MOGOLLÓN JAIMES

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO - MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA

Y DE TELECOMUNICACIONES

BUCARAMANGA

2013

ALGORITMO DE OPTIMIZACIÓN GOTAS DE AGUA VIRTUALES INTELIGENTES APLICADO A LA PLANEACIÓN DE RUTA ÓPTIMA DE UN ROBOT MÓVIL

JUAN HERMÓGENES ARIAS BARAJAS

MOISÉS FRANCISCO MOGOLLÓN JAIMES

Trabajo de grado para optar el título de INGENIERO ELECTRÓNICO

Director:

RODRIGO CORREA, Ph.D.

Codirector:

Ing. IVÁN MAURICIO AMAYA, Ph.D(c)

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FÍSICO - MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA

Y DE TELECOMUNICACIONES

BUCARAMANGA

2013

DEDICATORIA

En primer lugar agradezco a nuestro Dios por guiarme y darme fuerzas para lograr este triunfo, además por brindarme su compañía en los momentos difíciles de la vida.

A mi madre María del Carmen porque con su infinito amor, y comprensión lo ha dado todo para ayudarme a llegar a mis objetivos y a buscar mi felicidad.

A mi padre Juan Bautista quien con su amor y sabiduría me ha enseñado a superar los obstáculos.

A mis hermanos Belcy, Diocelina y Josue por su apoyo y comprensión.

A mis compañeros y amigos porque con ellos he compartido alegrías y tristezas, y en especial a Moisés por transmitirme sus conocimientos.

Juan.

DEDICATORIA

A Dios, por darme paciencia y voluntad en los momentos difíciles de mi vida.

A mi madre y hermano por confiar en mí, y siempre tener palabras de a aliento que me permitieron continuar.

A mi padre, que aunque no está presente en vida, su ejemplo hizo de mí, una persona persistente.

A mi esposa, por su comprensión y amor que me brinda día a día.

Finalmente a mis amigos, que nunca se cansaron en animarme, gracias a todos ellos por contribuir a construir mi vida y a culminar exitosamente con este proyecto de grado.

Moisés.

AGRADECIMIENTOS

Un agradecimiento especial al profesor Rodrigo Correa director del proyecto, por su sabiduría, comprensión y entrega durante todo el desarrollo del proyecto.

A Iván Amaya codirector del proyecto por su disposición y ayuda en este trabajo.

A todos los profesores por guiarnos y contribuir en nuestra formación como personas y profesionales.

A la Universidad Industrial de Santander por proporcionarnos las herramientas necesarias para culminar con éxito nuestras metas.

A nuestros compañeros y amigos por compartir sus conocimientos y alegrías.

TABLA DE CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

RESUMEN

TÍTULO: ALGORITMO DE OPTIMIZACIÓN GOTAS DE AGUA VIRTUALES INTELIGENTES APLICADO A LA PLANEACIÓN DE RUTA ÓPTIMA DE UN ROBOT MÓVIL[*].

AUTORES: JUAN HERMÓGENES ARIAS BARAJAS Y MOISÉS FRANCISCO MOGOLLÓN JAIMES[**].

PALABRAS CLAVE: Algoritmo gotas de agua inteligentes, TSP, laberintos prototipo.

Este trabajo contiene la descripción, implementación y validación del algoritmo gotas de agua virtuales inteligentes. Con base en los conceptos fundamentales sobre el funcionamiento del algoritmo, se implementó un código en MATLAB® que permite trazar rutas óptimas para robots móviles. Para la validación del algoritmo se utilizó la relación entre el problema del agente viajero (TSP) y la planificación de rutas, además de la solución de laberintos prototipo.

La verificación del algoritmo para algunas de las configuraciones de nodos, se realizó simulando condiciones similares a las que presentan los trabajos realizados por *Hamed Shah-Hosseini.* Los resultados obtenidos muestran que la solución de nodos distribuidos en un círculo mediante este algoritmo, presentan crecimientos abruptos en alrededor de 70 nodos y 100 nodos. También es de notar que el comportamiento del algoritmo es tipo exponencial, y no monotónico, esto se debe a que este algoritmo permite la aleatoriedad para resolver problemas combinatorios, como por ejemplo en la solución de configuraciones del problema TSP, en las que se obtienen resultados aproximados al exacto, para un tiempo de cómputo razonable a pesar de que éste crezca exponencialmente con el incremento en la dimensión del problema.

Con esta investigación se comprueba la capacidad del algoritmo para solucionar configuraciones clásicas del problema TSP, aún para situaciones donde un robot móvil requiere visitar más de 100 nodos. Además es evidente la eficiencia del algoritmo frente a la solución de laberintos en tiempo breve, ya que no requiere visitar todos los nodos.

[*] Proyecto de Grado

[**] Facultad de Ingenierías Físico- Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Ph.D. Rodrigo Correa. Codirector: Ph.D(c). Iván Amaya.

ABSTRACT

TITLE: OPTIMIZATION ALGORITHM INTELLIGENT VIRTUAL WATER DROPS APPLIED TO THE OPTIMAL PATH PLANNING FOR A MOBILE ROBOT[*].

AUTHORS: JUAN HERMÓGENES ARIAS BARAJAS AND MOISÉS FRANCISCO MOGOLLÓN JAIMES[**].

KEYWORDS: Intelligent water drops algorithm, TSP, prototype mazes.

This work contains the description, implementation and validation of the algorithm intelligent virtual water drops. Based on the fundamental concepts on the algorithm performance, it was implemented in MATLAB® a code that allows to draw optimal paths for mobile robots. For validation of the algorithm it was used the relationship between the traveling salesman problem (TSP) and route planning, in addition to the solution of mazes prototype.

Verification of the algorithm to some node configurations were conducted simulating conditions similar to those of the work performed by*Hamed Shah-Hosseini.* The results show that the solution of distributed nodes in a circle using this algorithm, show abrupt increases around 70 nodes and 100 nodes. Also, it is noteworthy that the algorithm behavior is of exponential kind, and not monotonic, this is because this algorithm allows the randomness to solve the combinatorial problems, such as in the solution of TSP problem configurations in which results obtained are approximate to exact, for a reasonable computation time even though it grows exponentially with the increase in the dimension of the problem.

This research tests the ability of the algorithm to solve classical configurations of TSP problem, even for situations where a mobile robot needs to visit more than 100 nodes. It is also evident the efficiency of the algorithm versus the solving mazes at short time because they do not need to visit all nodes.

[*] Degree Draft

[**] Facultad de Ingenierías Físico- Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Ph.D. Rodrigo Correa. Codirector: Ph.D(c). Iván Amaya.

INTRODUCCIÓN

Los algoritmos de optimización se han convertido en una alternativa en la planificación de rutas para robots móviles, ya que estos permiten acercarse a la ruta óptima en aplicaciones tales como la minería, la exploración, la navegación, en búsqueda de personas, entre otras. Para estas situaciones, hallar la ruta exacta se convierte en un problema complejo, en donde su mayor dificultad es el incremento en el número de caminos que puedan hacer parte de dicha ruta.

El presente trabajo de grado en la modalidad de investigación, toma como ejemplo de aplicación, la planeación de rutas para un robot móvil mediante el algoritmo gotas de agua inteligentes[1] o *IWD (Intelligent Water Drops)* [11], que permite encontrar la ruta óptima que debe seguir un robot, a través de una serie de puntos distribuidos aleatoriamente.

Este informe inicia con la transcripción del problema y objetivos aprobados para su desarrollo. En el capítulo 2, se hace una descripción de los principales aspectos fundamentales, como el algoritmo gotas de agua virtuales inteligentes, y el problema del agente viajero o TSP (*Traveling Salesman Problem)*. Posteriormente en el capítulo 3, se describe la implementación de algoritmo, en donde se explican las variables utilizadas y el diagrama flujo correspondiente. En el capítulo 4, se presentan los experimentos y análisis de resultados para dos condiciones: la primera, probar el algoritmo con un número de nodos distribuidos en un círculo y algunas configuraciones del problema TSP, y la segunda, para solucionar laberintos prototipo. En la sección 4.5 se describe la interfaz gráfica y finalmente en el capítulos 5 se presentan las conclusiones, y en el capítulo 6 las recomendaciones y trabajo futuro.

[1] En este documento se llamarán gotas de agua virtuales inteligentes, porque son un símil de las gotas de agua naturales.

1. DESCRIPCIÓN DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA[2]

En el presente trabajo de grado se propone desarrollar el problema consistente en determinar la ruta óptima de desplazamiento bidimensional de un robot móvil, utilizando el algoritmo de gotas de agua virtuales inteligentes [11]. A título demostrativo, se resolverán unos laberintos en que pudiera verse envuelto el robot durante su desplazamiento. Esta aplicación no se ha reportado en la literatura mediante la estrategia planteada en este trabajo, y constituye una contribución al tema.

A continuación se transcriben, los objetivos aprobados para el desarrollo del presente trabajo de grado, en la modalidad de investigación.

1.2 OBJETIVOS

1.2.1 Objetivo general[3]

Proponer y demostrar, mediante una aplicación en particular, el uso del algoritmo gotas de agua virtuales inteligentes, en el área de planificación de rutas de robots móviles.

1.2.2 Objetivos específicos[2]

1. Implementar en una herramienta computacional como MATLAB^{MR}, el algoritmo gotas de agua virtuales inteligentes, (página 27)

[2] Es un aporte intelectual colectivo y consensuado de los estudiantes Arias y Mogollón, junto con el profesor Rodrigo Correa, e Iván Amaya.

[3] Este texto es un aporte intelectual de los estudiantes Arias y Mogollón, y del profesor Rodrigo Correa, proponente del tema de este trabajo de grado y que hace parte integral de la línea de investigación en *Optimización* reconocida dentro del grupo CEMOS (categoría A1).

2. Aplicar el algoritmo gotas de agua virtuales inteligentes para planear trayectorias de robots móviles, para dos condiciones: la primera en la identificación de la trayectoria más corta que enlace un conjunto de puntos de interés, sin obstáculos, sin repetirse y retornando a su posición inicial. La segunda en la solución de laberintos, (página 33)

3. Realizar un análisis estadístico básico de los resultados obtenidos, (página 38)

4. Programar una interfaz gráfica para usuario que permita visualizar el funcionamiento del algoritmo, (página 55).

2. MARCO TEÓRICO

2.1 CONCEPTOS PRELIMINARES

Los algoritmos de optimización permiten encontrar la mejor solución a un problema, ya sea en términos de costos, tiempo u otro parámetro que se esté considerando. Estos se describen mediante una función objetivo, la cual relaciona los parámetros de entrada, con los de salida, y en algunos casos depende de restricciones particulares de la situación que se esté analizando.

Entre estos tipos de algoritmos, se encuentran los exactos y los aproximados. Los exactos o también llamados determinísticos, tratan de encontrar la solución óptima e inmejorable de un problema a costa de un gasto de tiempo de cómputo relativamente alto e inaceptable [16], ya que garantizan que todo el conjunto de soluciones factibles ha sido considerado; además para la misma entrada, siempre se obtiene la misma salida [1], igualmente estos desarrollan el problema mediante un algoritmo definido sin lugar a la aleatoriedad; este incluye muchas veces características como las derivadas de la función objetivo.

Por otra parte, están los algoritmos aproximados, que se utilizan en aplicaciones donde el tiempo disponible para encontrar una solución debe ser razonable [16]. Estos permiten hallar una solución aproximada, que cumpla con las necesidades del problema y que esté en un intervalo válido de valores. Esta clase de algoritmos de optimización producen salidas diferentes para la misma entrada [16], pero con beneficios muy cercanos. Además, solo consideran las regiones del espacio de solución con mayor probabilidad de encontrar el resultado adecuado, reduciendo así la cantidad de tiempo empleado para la solución del problema [1].

Los algoritmos de aproximación se pueden clasificar en heurísticos y metaheurísticos. Los primeros son de carácter específico para cada problema, mientras que los segundos utilizan procedimientos más generales. En algunas aplicaciones, los métodos heurísticos caen en óptimos locales, y no poseen mecanismos para escapar de ellos. Por lo tanto es conveniente utilizar los algoritmos metaheurísticos, los cuales guían a los heurísticos de tal manera, que realicen una búsqueda más allá de los óptimos locales [9].

Dentro de este último grupo se encuentran los algoritmos inspirados en los sistemas naturales, que tienen la capacidad de describir y resolver problemas complejos de manera óptima a partir de condiciones iníciales muy sencillas pero con resultados asombrosos. El principio básico de estos algoritmos, es que poseen características similares a las que tienen los sistemas naturales que se están describiendo. En la actualidad se modelan fenómenos naturales tales como, las relaciones entre los seres vivos, las redes neuronales, la biología celular y molecular, el sistema inmune, los sistemas de enjambres, entre otros.

2.2 EL ALGORITMO GOTAS DE AGUA VIRTUALES INTELIGENTES

El algoritmo gotas de agua inteligentes o *Intelligent Water Drops* (IWD) hace parte de un grupo de algoritmos inspirados en los procesos naturales. Fue desarrollado

por el investigador iraní *H. Sahah-Hosseini* en el año 2007 [11], quien se basó en la descripción del proceso natural del flujo del agua de los ríos, modelado a partir de la interacción entre las gotas de agua y el suelo del lecho del rio [11, 12, 7]. La idea básica es tomada de las gotas de agua naturales individuales, que presentan un comportamiento particular en la forma en que fluye el río.

2.2.1 Descripción cualitativa del modelo

Para comprender el algoritmo IWD, se puede tomar como referencia la observación y descripción del comportamiento de las gotas de agua naturales, desde que inician su recorrido hasta que finalmente llegan a su destino, el cual puede ser un lago, o un mar. Este algoritmo es un símil de tal situación física y de ninguna manera representa la realidad en forma exacta, ni lo intenta.

Suponiendo que una gota de agua por alguna causa, se encuentra localizada en la parte más alta de una colina, esta empieza a moverse a una velocidad específica debido a la fuerza de gravedad que la atrae hacia el centro de la tierra y a la inclinación del terreno. La composición química y física de la gota de agua le permite interactuar con su entorno, de manera que puede deslizarse a través de cualquier terreno, y en su recorrido, se une con otras gotas y poco a poco forman un río, el cual se puede considerar como un enjambre de gotas de agua.

La trayectoria ideal que debería escoger un río, desde su fuente hasta su destino final sería una línea recta, sin embargo en la realidad el río sigue un recorrido con muchas curvas, esto se debe a los diferentes obstáculos que hay en la naturaleza, como los son montañas, rocas, tierra, árboles, entre otros. En su recorrido, las gotas de agua se encuentran con diferentes caminos cada uno con características particulares; por lo tanto, las gotas deben seleccionar uno de estos, y por lo general siempre escogen la ruta óptima, que podría ser la que represente la menor distancia, y menor cantidad de obstáculos.

2.2.1.1 Propiedades de las gotas de agua

Las gotas de agua en todo momento tratan de cambiar la ruta real, para convertirla en la óptima [11]. Esto se ve claramente cuando un río arrastra cierta cantidad de tierra de un lugar a otro, o cuando se desborda buscando llegar más rápido a su destino. Además, las características del terreno pueden cambiar el curso del río, también su velocidad y la cantidad de sedimentos que transporta.

De lo anterior se pueden deducir dos características de las gotas de agua que son de especial atención para el modelo que se está analizando: una de estas es la velocidad propia de cada gota y la otra es la cantidad de tierra o suelo que es capaz de transportar, la cual es transferida de las partes más rápidas a las más lentas del entorno [11, 12, 4].

Las gotas de agua tienen la capacidad de modificar estas dos propiedades a medida que cambia el entorno natural, es decir, que buscan adaptarse cuando pasan de un entorno a otro.

2.2.1.2 Gotas de agua inteligentes

Tomando como semejanza las gotas de agua naturales, se desarrolló un método de solución de problemas llamado gotas de agua virtuales inteligentes [11]. Su nombre se refiere, a que las gotas de agua naturales actúan en su entorno como si tuviesen “mente propia” para encontrar la ruta más fácil, desde un origen hasta un destino desconocido.

Inicialmente en este método se puede considerar, que cada gota se mueve en pasos de longitud discretas, a través de cierto número de puntos llamados nodos, empezando aleatoriamente en uno de ellos y luego gracias a sus propiedades, decide cual es el próximo nodo para visitar, cumpliendo su recorrido a través de

todos los nodos que corresponden a un problema en particular. Hay que tener en cuenta que el mismo nodo no sea visitado dos veces por la misma gota.

Una gota de agua en su recorrido siempre adquiere cierta cantidad de tierra, que incrementa su peso y por ende su velocidad, también causa que la cantidad de suelo entre los nodos disminuya, ocasionando que las próximas gotas de agua escojan las rutas que poseen menos cantidad de tierra, facilitando así su movimiento. Este fenómeno se modela mediante una distribución aleatoria uniforme, de manera tal que asigne un valor determinado a la cantidad de suelo entre enlaces de cada par de nodos.

2.2.2 Modelo analítico del algoritmo IWD [11, 12, 5]

El algoritmo de gotas de agua virtuales inteligentes, presenta dos propiedades importantes obtenidas de una gota de agua natural:

- La cantidad de suelo que transporta, denotado por *soil(IWD)*.
- La velocidad que esta posee, denotada por *velocity(IWD).*

Suponiendo que una gota de agua inteligente se mueve en pasos de longitud discretas, desde un punto i hasta otro punto j, su velocidad se incrementa en una cantidad definida por $\Delta vel^{IWD}(t)$, que es inversamente proporcional al suelo que existe entre los dos nodos, es decir que si la gota de agua reduce el suelo del lecho del rio llevándose cierta cantidad de tierra, causa un cambio en su velocidad. Este comportamiento se modela mediante la siguiente ecuación:

$$\Delta vel^{IWD}(t) = \frac{a_v}{b_v + c_v * soil(i,j)} \qquad (1)$$

donde a_v, b_v, c_v son parámetros positivos seleccionados por el usuario, y $soil(i,j)$ es la cantidad de tierra que hay entre los nodos i y j, el parámetro b_v evita que la ecuación anterior se vuelva indeterminada cuando $soil(i,j)$ sea cero, causando

que el algoritmo no funcione correctamente. Este caso sucede cuando la gota de agua retorna al punto de partida.

Por otro lado, la cantidad de tierra aumenta de manera no lineal por el incremento de su velocidad, y se relaciona con el tiempo que tarda de ir de un nodo a otro, se tiene entonces que:

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s * time(i,j;vel^{IWD})} \quad (2)$$

En esta ecuación los parámetros $a_s, b_s, y\, c_s$ son números positivos, también definidos por el usuario. Al igual que en la ecuación de velocidad de la gota de agua, en este caso el parámetro b_s, también evita una división por cero.

De la definición clásica para la velocidad lineal, el tiempo es inversamente proporcional a la velocidad de la gota de agua y proporcional a la distancia. Entonces el tiempo que tarda una gota de agua para desplazarse entre los dos nodos se determina así:

$$time(i,j;vel^{IWD}) = \frac{HUD(i,j)}{máx(\varepsilon_v, vel^{IWD})} \quad (3)$$

$HUD(i,j)$ es una función heurística local, definida dependiendo del problema en particular, la cual mide la no deseabilidad de una gota de agua para moverse de un nodo a otro. El término $máx(\varepsilon_v, vel^{IWD})$ significa que se debe seleccionar la velocidad máxima de la gota, y para el caso donde la velocidad sea cero, se escoge ε_v para evitar una indeterminación.

Como se había mencionado, la cantidad de suelo entre dos nodos se ve afectado por la acción que realiza la gota al camino que los une, por lo tanto se tiene la siguiente expresión:

$$soil(i,j) = \rho_o * soil(i,j) - \rho_n * \Delta soil(i,j) \quad (4)$$

Aquí ρ_o y ρ_n son un números entre 0 y 1. Es de notar que $\Delta soil(i,j)$ es la cantidad de tierra que pierde el camino, pero es ganada por la gota de agua. La cantidad de suelo que gana la gota de agua se expresa como:

$$soil^{IWD} = soil^{IWD} + \Delta soil(i,j) \quad (5)$$

Adicionalmente, otro comportamiento modelado es la preferencia de la gota de agua por los caminos con menos obstáculos; esto se implementa mediante una distribución uniforme, que establece la probabilidad que una gota elija un camino u otro. Esta probabilidad es inversamente proporcional a la cantidad de suelo entre los dos nodos i y j; esto significa que cuanto menor sea la cantidad de suelo en un camino, más posibilidad tiene este camino para ser seleccionado. La función de probabilidad debe cumplir las siguientes propiedades:

- $f(x) \geq 0, para\ todo\ x$
- $\sum f(x) = 1.$

La ecuación que determina la probabilidad se expresa de la siguiente manera:

$$p(i,j;IWD) = \frac{f(soil(i,j))}{\sum_{k \notin N(IWD)} f(soil(i,k))} \quad (6)$$

dónde $N(IWD)$ es el conjunto de nodos que han sido visitados por la gota; la expresión $f(soil(i,j))$ es una función que depende de la cantidad de suelo que hay entre dos nodos y está dada así:

$$f(soil(i,j)) = \frac{1}{\varepsilon_s + g(soil(i,j))} \quad (7)$$

La constante ε_s es un número positivo pequeño que impide otra posible indeterminación del algoritmo. El término $g(soil(i,j))$ se utiliza para desplazar la cantidad de suelo entre i y j, y se expresa de la siguiente forma:

$$g(soil(i,j)) = \begin{Bmatrix} soil(i,j) & if\ min_{l \notin vc(IWD)}(soil(i,l)) \geq 0 \\ soil(i,j) - min_{l \notin vc(IWD)}\big(soil(i,l)\big) & otro\ caso \end{Bmatrix} \quad (8)$$

El término $min_{l \notin vc(IWD)}$ devuelve el menor valor del argumento, y por lo tanto el algoritmo selecciona $soil(i,j)$ cuando $min_{l \notin vc(IWD)}(soil(i,l))$ es mayor o igual a cero. Para los casos donde el valor del argumento es menor que cero el algoritmo escoge la opción $soil(i,j) - min_{l \notin vc(IWD)}\big(soil(i,l)\big)$, de modo tal que esta expresión tiene como objetivo volver los valores negativos en positivos ya que las longitudes reales son positivas.

2.3 PROBLEMA DEL VENDEDOR VIAJERO Y EL ALGORITMO GOTAS DE AGUA INTELIGENTES

El problema clásico del agente viajero o TSP (*Traveling Salesman Problem)*, es un problema de optimización combinatoria, y tiene diferentes aplicaciones en ingeniería y en problemas relacionados con la planificación de rutas en general. La situación se trata de que un vendedor de mercancías debe recorrer cierto número de ciudades ubicadas en diferentes puntos geográficos, y tiene como objetivo encontrar la mínima ruta entre todos los recorridos posibles [11, 4]. Para lograr este propósito debe cumplir con una serie de reglas las cuales son:

- visitar cada ciudad solo una vez
- Debe escoger la ruta más corta para cada situación
- Debe entregar cierta cantidad de mercancía en cada ciudad
- Llegar al punto de partida nuevamente

Para solucionar el problema del agente viajero o TSP, con el algoritmo IWD, inicialmente se debe modelar el problema del TSP a través de un gráfico o mapa adecuado, y específico para cada problema en particular, en donde se puedan representar los nodos con sus coordenadas (N, p). En nuestro caso los nodos

N son las ciudades, y p son sus posiciones. Luego se deben representar los enlaces entre las ciudades, a los que se les proporciona un peso, que indica la cantidad de tierra del enlace. Finalmente como la gota solo puede visitar una ciudad, se implementa una lista de ciudades ya visitadas, de tal manera que la gota no vuelva a visitar la misma ciudad dos veces [11, 12, 4]. En este caso la distancia que hay entre dos ciudades i y j, representa la **función objetivo** definida como la distancia euclidiana, que significa el segmento de la recta entre los extremos de los dos puntos.

Otro enfoque para desarrollar el problema del agente viajero es tratar las gráficas mediante el ciclo Hamiltoniano, en este caso, durante un ciclo la gota solo puede visitar una sola vez el vértice o nodo de la gráfica que representa la ubicación de la ciudad. En este tipo de solución se encuentran dos clases de TSP. Una es la forma simétrica (STSP), en la que sólo hay un camino entre dos ciudades i y j, siendo igual el recorrido de i hasta j como a la inversa. La STSP es la más utilizada para verificar los algoritmos. La otra clase es la asimétrica (ATSP) en la cual pueden haber dos caminos para llegar a las dos ciudades [4].

La dificultad para resolver el problema TSP, radica en que los algoritmos que hay para solucionarlo, requieren de tiempos de cómputo que crecen exponencialmente con el tamaño del problema. Para mostrar esto veamos un ejemplo: supongamos que se tienen N ciudades, su solución implica calcular las rutas para cada una de las combinaciones y luego se selecciona la ruta más corta, el tiempo de ejecución para este algoritmo sería el tiempo empleado para revisar $f(n) = n!$ caminos y de todos ellos, seleccionar la mejor [3].

La Tabla 1 muestra las diferencias de crecimiento de algunas funciones de tiempo, (las cifras que se muestran son tiempo de procesamiento en una computadora que procesa 1 millón de operaciones de punto flotante por segundo) tomada de [3].

Tabla 1. Comparación de varias funciones exponenciales de tiempo de cómputo.

	Tiempo de cómputo $t(f(n))$	
Tamaño n	$t(2^n) = \frac{2^n}{1 * 10^6}$	$t(3^n) = \frac{3^n}{1 * 10^6}$
10	$0.001[s]$	$0.059[s]$
20	$1.0[s]$	$58[minutos]$
30	$17.9[minutos]$	$6.5[años]$
40	$12.7[días]$	$3855[siglos]$
50	$35.7[años]$	$2 * 10^8[síglos]$
60	$366[síglos]$	$1.3 * 10^{13}[síglos]$

Fuente: [3].

El problema TSP es importante porque permite adaptar su modelo a muchos problemas reales, con características diferentes pero el mismo principio básico. A estos modelos de nodos se les conoce como configuraciones de la TSP. En nuestro caso son utilizadas para validar el algoritmo en la planeación de rutas para un robot móvil. Las configuraciones clásicas de la TSP se utilizan comúnmente para probar el funcionamiento de diferentes tipos de algoritmos de optimización, de tal modo que se puedan comparar resultados obtenidos ya sea con otros algoritmos, o también mediante métodos exactos. Entre algunas de estas configuraciones están la eil51, Berlin52, st70, eil76, pr76, KroA100, Bier176.

En la Tabla 2 se muestran los mejores valores conocidos para encontrar la mejor ruta de dichas configuraciones, donde también se presentan el número de ciudades para cada una de ellas [8]. Es de notar que los datos de los valores conocidos no tienen unidades, ya que son solo una representación de un problema, por ejemplo: para una situación particular esos valores pueden ser distancias, en [cm], [m] o [km].

Tabla 2. Algunas configuraciones utilizadas con la TSP.

IDENTIFICADOR	CONFIGURACIÓN	MEJOR VALOR CONOCIDO
Eil51	Eil51(51 ciudades)	426
Berlin52	Berlin52(52 ciudades)	7542
St70	St70(70 ciudades)	675
Eil76	Eil76(76 ciudades)	538
KroA100	KroA100(100 ciudades)	21282

Fuente: [8].

3. IMPLEMENTACIÓN DEL ALGORITMO GOTAS DE AGUA VIRTUALES INTELIGENTES

El algoritmo gotas de agua virtuales inteligentes, fue programado sobre la herramienta computacional MATLAB®, este es aplicado a la planeación de rutas de robots móviles. Su validación se realizó solucionando un conjunto de configuraciones propuestas en la TSPLIB (disponible en internet), y algunas de estas solucionadas por *Hamed Sahah-Hosseini* [11, 12, 7], para esto se buscaron reproducir los resultados bajo las mismas condiciones de prueba, esto con el fin de comparar los resultados obtenidos, con los que presenta el Sahah-Hosseini. Además de esto la solución del problema TSP mediante el algoritmo IWD, es similar a la de otros problemas como lo son el ruteo de vehículos VRP [6], de aviones no tripulados UCAV [5], y de despacho de mercancías [10]. En la Tabla 3, se presenta el conjunto de variables y constantes utilizadas en la programación del algoritmo, en la cual se describe el significado y el tipo (es decir si es entero, matriz o vector).

Tabla 3. Conjunto de variables y constantes utilizadas en el algoritmo.

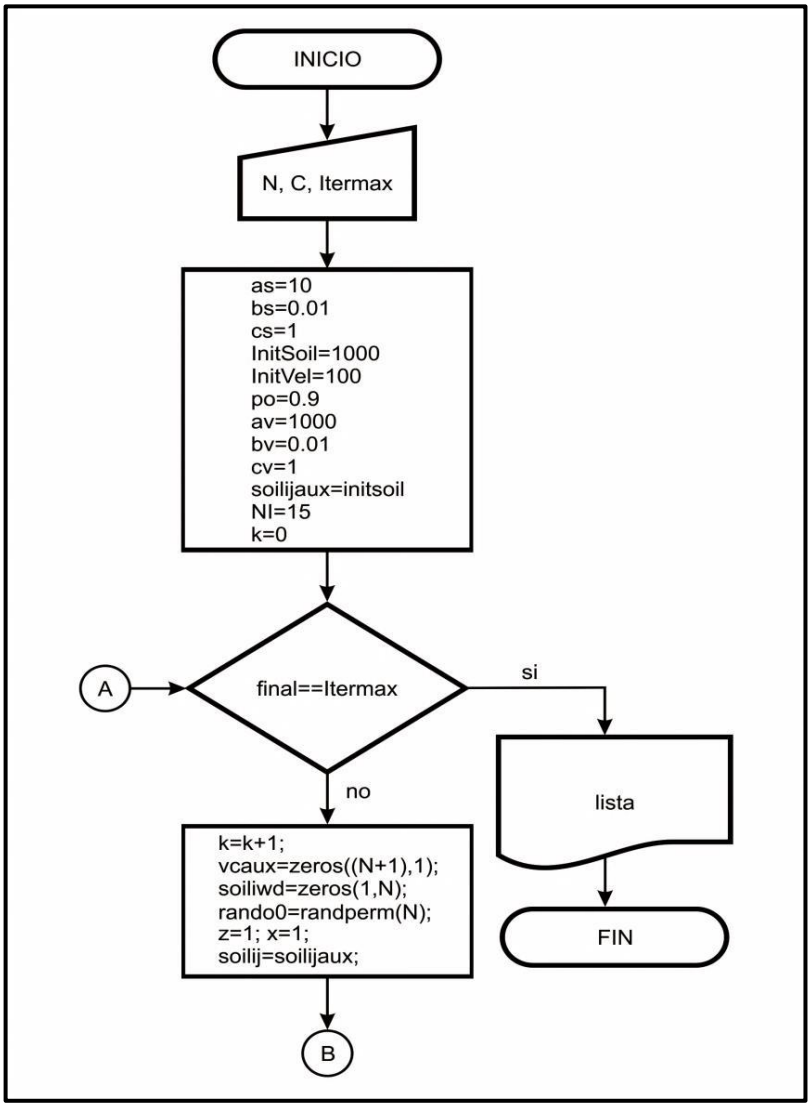
PARÁMETRO	DESCRIPCIÓN	TIPO
N	Número de gotas	Entero
Itermax	Cantidad máxima de iteraciones a realizar	Entero
C	Coordenadas de los nodos	Matriz Nx2
as, bs, cs	Parámetros de la actualización del suelo	Enteros
av, bv, cv	Parámetros de la actualización de la velocidad	Enteros
rando0	Vector de permutaciones aleatorias	Vector 1xN
Initsoil	Suelo inicial	Entero
Initvel	Velocidad inicial	Entero
Veliwd	Velocidad de la gota	Vector 1xN
Soilij	Suelo local entre los enlaces	Matriz NxN
Soilijaux	Suelo global entre los enlaces	Matriz NxN
Po	Parámetro para actualizar el suelo local y global	Entero
Z	Dirección para los enlaces	Entero
X	Dirección para las gotas	Entero
Leng	Longitud del recorrido, inicialmente infinito	Vector
Vc	Máscara de ciudades visitadas	Vector Nx1
Vcaux	Listado de coordenadas de ciudades visitadas	Vector (N+1)x1
HUD	Longitud entre nodos, distancia euclidiana	Vector Nx1
Time	Tiempo que tarda la gota	Entero
Deltasoil	Cantidad de suelo que arrastra la gota	Entero
Tb	Longitud total de la gira (recorrido)	Entero
Tm	Longitud mínima entre giras	Entero
C	Identificador de longitud mínima	Entero
NI	Parámetro para reinicializar el algoritmo	Entero
Inicial	Coordenada inicial	Entero
Próximo	Coordenada próxima	Entero
i y j	Coordenadas de ruta óptima	Vector 1xN
Lista	Puntos indexados de los enlaces para la ruta óptima	Vector 1x(1+N)
Final	Contador para cumplir la condición de terminación	Entero
K	Contador para terminar el proceso	Entero

Fuente: Autores.

El código fue diseñado, para que el usuario ingrese las coordenadas del conjunto de nodos a visitar y el número máximo de iteraciones a realizar. Por otra parte, el programa arroja un vector de direcciones que representa los enlaces con la solución encontrada, este vector llamado lista, se encuentra indexando las coordenadas de los nodos, es decir que tiene un vector que ordena los números

del 1 al N, de tal manera que represente los enlaces con la ruta más corta, para después leer estos datos y representarlos con sus respectivas coordenadas.

Figura 1. Primera parte del algoritmo.



Fuente: Autores.

El diagrama del algoritmo se presenta en tres partes conectadas a través de los puntos A, B y C. La primera corresponde a la Figura 1, que muestran los datos entrada y salida, la inicialización de los parámetros estáticos, y dinámicos, estos últimos corresponde al listado de ciudades visitadas, al suelo que arrastra cada

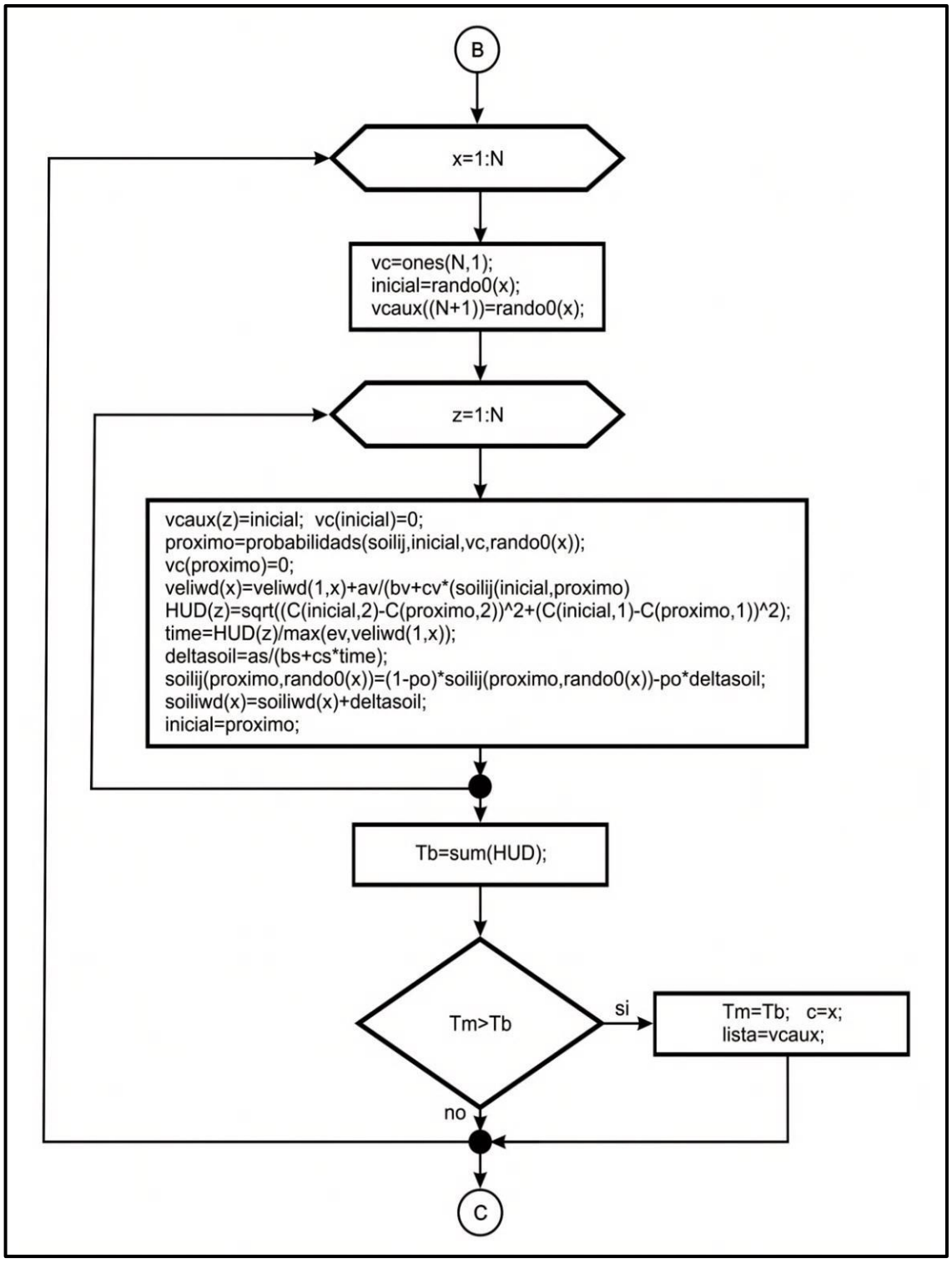
gota de agua, al contador k, al suelo entre los enlaces y la generación de la variable rando0, que se utiliza para la distribución aleatoria de las gotas de agua, sobre las direcciones de cada nodo en cada iteración.

La segunda parte del algoritmo, se observa en la Figura 2, esta corresponde a dos estructuras *for* anilladas, que permiten realizar el recorrido por los nodos para cada una de las gotas asignadas y luego guardar los registros de interés, tales como el tiempo que tarda y el suelo que arrastra cada gota. Por otra parte estas estructuras sirven para desplazarse por la matriz de los suelos locales. La variable inicio, representa la posición actual de la gota y la variable próxima, el siguiente nodo al que debe desplazarse; este es encontrado utilizando una función de probabilidad, donde al existir menos cantidad de suelo entre los enlaces aumenta su posibilidad de escoger un nodo específico. Esta variable inicio es programada acorde a las ecuaciones (5), (6) y (7).

En la ecuación (8) se observan dos condiciones, esto permite desplazar los valores negativos de los suelos si se presentan en alguna iteración. Conociendo el próximo nodo, se actualiza el listado de ciudades visitadas y la velocidad de la gota de agua. Posteriormente se calcula la distancia entre los nodos para poder determinar el tiempo que tarda la gota y la cantidad de suelo (tierra) que ella arrastra. Conociendo el suelo que lleva la gota, se debe actualizar el nuevo suelo que existe entre los enlaces y la cantidad de suelo que lleva actualmente la gota.

Finalmente la variable inicial se ubica en la variable próxima, para iniciar nuevamente otro ciclo que se repetirá para cada gota, hasta terminar cada una de ellas su respectivo recorrido. La variable Tm se actualiza dependiendo si el recorrido actual es menor al anterior; cuando esto ocurre, se identifica cuál de las gotas fue la que realizó tal recorrido y se guarda en la variable lista el orden o secuencia de los nodos visitados.

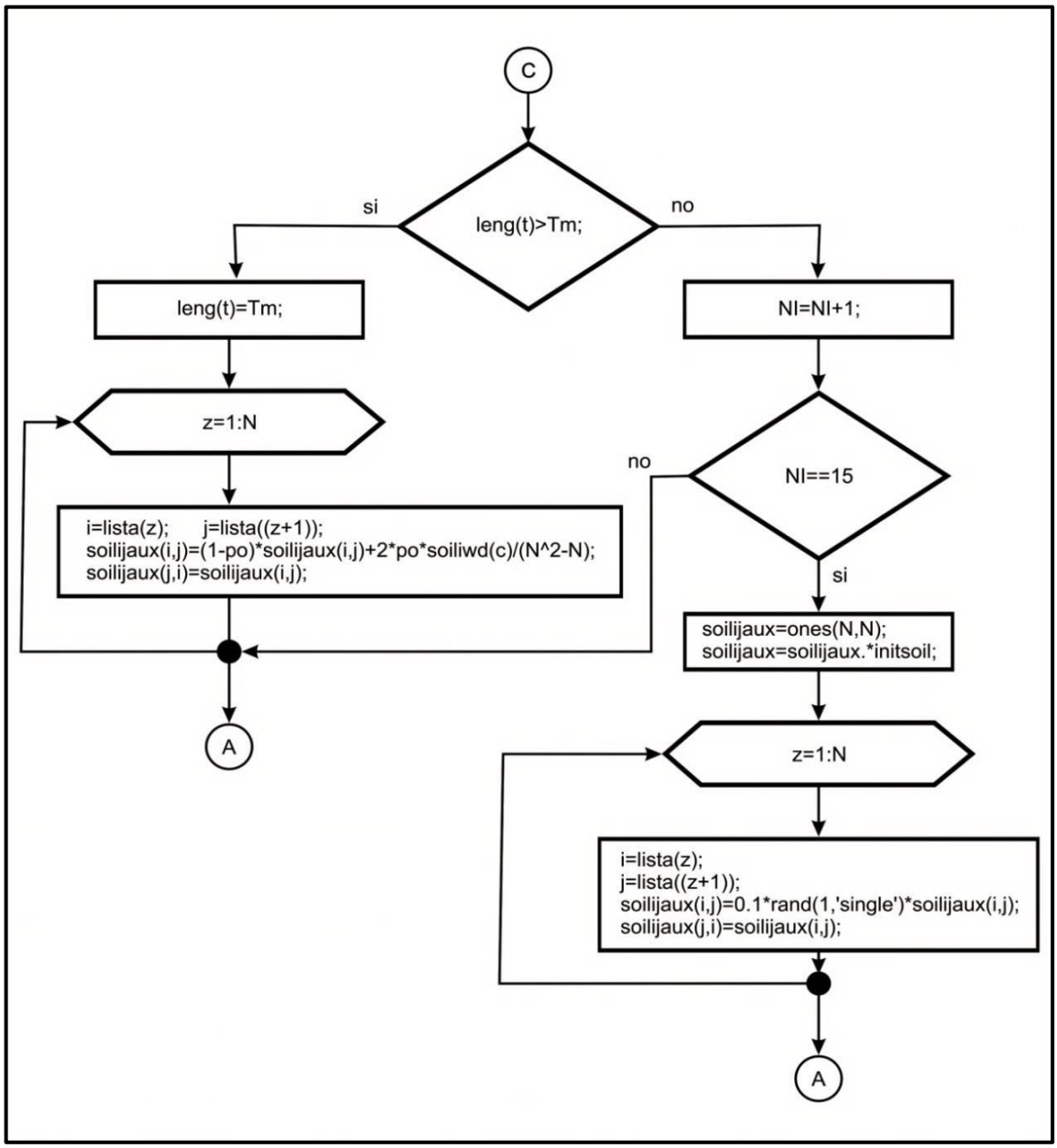
Figura 2. Segunda parte del algoritmo.



Fuente: Autores.

Al terminar los NxN ciclos, se continúa con la tercera parte del algoritmo, que se observa en la Figura 3. En este momento pueden ocurrir dos eventos, uno es que la longitud actual del recorrido no sea mayor a la encontrada en la iteración anterior y el otro es el caso contrario.

Figura 3. Tercera parte del algoritmo.



Fuente: Autores.

Cuando se presenta el primer evento, se lleva a cabo la reinicialización de los suelos entre los enlaces, únicamente cuando este evento haya ocurrido 15 veces. Si ocurre el segundo evento, se procede a la actualización de la longitud del recorrido con la variable leng y a la actualización del suelo global soilijaux, esto permite mejorar las probabilidades de éxito, con la esperanza de evitar los óptimos locales, para que en la siguiente iteración se seleccione el recorrido mínimo hallado. Finalmente se pregunta si ya se cumplió el número de total de iteraciones proporcionadas por el usuario.

4. EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

4.1 ENTORNO COMPUTACIONAL

Todos los algoritmos fueron programados en MATLAB® y las ejecuciones se desarrollaron en un computador *Hewlett Packard (HP), ProBook 4320s,* Intel Core i3, de 2.4 GHz, con 2 GB de RAM, bajo el sistema operativo WINDOWS 7 Profesional para 32 bits.

4.2 DESCRIPCIÓN DE LAS ACTIVIDADES

Se han propuesto y desarrollado dos actividades para demostrar la validez del algoritmo gotas de agua virtuales inteligentes, como una alternativa de solución al problema de planificación de rutas para robots móviles. La primera consiste en comprobar el desempeño del algoritmo, utilizándolo para encontrar la ruta óptima de algunas distribuciones de nodos del problema TSP, las cuales se usan frecuentemente para hallar la eficiencia de diferentes algoritmos. La segunda actividad se trata de utilizar el algoritmo para encontrar la ruta más corta de algunos laberintos, los cuales simulan rutinas en las que pueda verse envuelto el robot durante su desplazamiento.

4.3 PRIMERA ACTIVIDAD: SIMULACIÓN DE RUTAS ÓPTIMAS PARA CONFIGURACIONES DEL PROBLEMA TSP

En esta actividad se pone a prueba el desempeño del algoritmo, solucionando algunas configuraciones del problema clásico TSP. El algoritmo tiene la capacidad de hallar el mejor recorrido, de una cierta cantidad de nodos que no presentan obstáculos, sin repetirse y retornando a su posición inicial, que corresponde al problema básico de TSP sin tomar en cuenta la variable de carga dejada, características cinemáticas del robot móvil ni obstáculos.

Se ejecutó el algoritmo para encontrar la ruta óptima para una distribución de ciudades alrededor de un círculo y las configuraciones eil51, berlin52, st70, eil76, y kroA100 tomadas de la librería TSPLIB (ver fuente [15]).

El entorno de solución es descrito por una gráfica que representa cada una de las configuraciones con sus respectivas coordenadas. El algoritmo que se utiliza en esta actividad es el que se presentó en el capítulo 2. En este caso se realizaron las pruebas tomando como referencia, condiciones similares a las que presentan los artículos [11, 12, 7, 13, 14] sobre algoritmo gotas inteligentes IWD.

Para cada una de las topologías de nodos nombradas anteriormente, se tomaron los datos longitud de la ruta óptima, número de iteraciones y tiempo. Debido a que el algoritmo *IWD* es estocástico, se obtienen valores óptimos diferentes dependiendo la cantidad de iteraciones que se efectúen para cada prueba, por lo tanto, el error está relacionado con el tiempo de cómputo para cada una de las configuraciones. El error es obtenido tomando como referencia los valores óptimos de las configuraciones que se encuentran relacionados en la Tabla 2, y en la fuente [15] .

4.3.1 Validación con nodos distribuidos formando un círculo

La validación del algoritmo se desarrolla a partir de la distribución de ciudades representadas por nodos sobre el perímetro de una circunferencia. Esto facilita identificar el recorrido óptimo que corresponde a la longitud del perímetro. Tomando un radio de 200 se obtiene la longitud máxima del recorrido que equivale a 1256.64, que es el valor óptimo a medida que existan tantos nodos en el perímetro que su cercanía describa una circunferencia suave sobre el perímetro del círculo. Se desarrolla la validación para una cantidad de hasta 100 nodos en intervalos de 10 nodos, empleando como condición de parada el valor máximo permitido del recorrido.

Los resultados encontrados para la validación del algoritmo, con nodos distribuidos uniformemente en una circunferencia se muestran en la Tabla 4.

Tabla 4. Resultados encontrados para la validación del algoritmo.

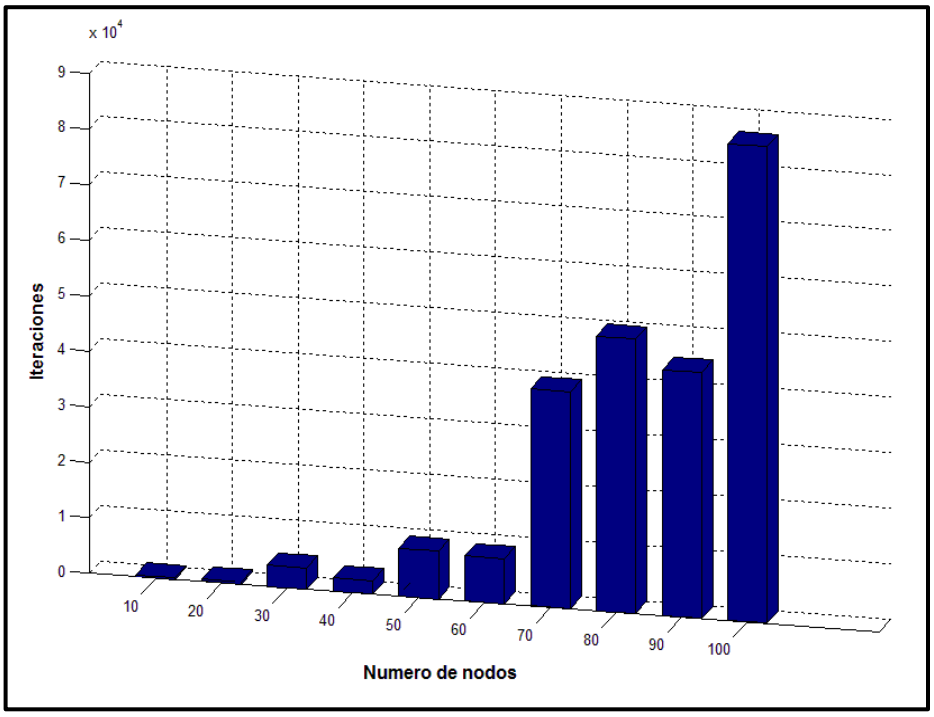
Nodos	10	20	30	40	50	60	70	80	90	100
Iteraciones	303	439	3797	2397	8645	8111	39080	49361	44275	85791
Tiempo [h]	0.0003	0.0015	0.03	0.0312	0.18	0.246	1.63	2.71	3.13	7.46

Fuente: Autores.

Esta validación permite describir de manera general el comportamiento del algoritmo con una distribución de nodos simétrica. Para otro tipo de configuraciones se presenta la misma tendencia exponencial en cuanto al incremento en el tiempo y el número de iteraciones respecto al incremento del número de nodos. La Figura 4 presenta los resultados encontrados después de ejecutar el algoritmo con la distribución de nodos anteriormente descrita. Se observa que a medida que se incrementa el número de nodos, también los hace el número de iteraciones; esto se debe al aumento del espacio de soluciones para resolver el problema. Además, para hallar el valor óptimo, el tiempo requerido para encontrar la solución adecuada también aumenta como era de esperarse, a

pesar que corresponda a la misma circunferencia. Igualmente se puede ver que el incremento no tiene un comportamiento monotónico y esto es debido a la naturaleza no lineal del algoritmo, es decir, que su comportamiento es aleatorio, debido a que las gotas de agua realizan un recorrido totalmente libre para cada iteración.

Figura 4. Gráfica de Iteraciones Vs Número de nodos, para una distribución de nodos en una circunferencia.

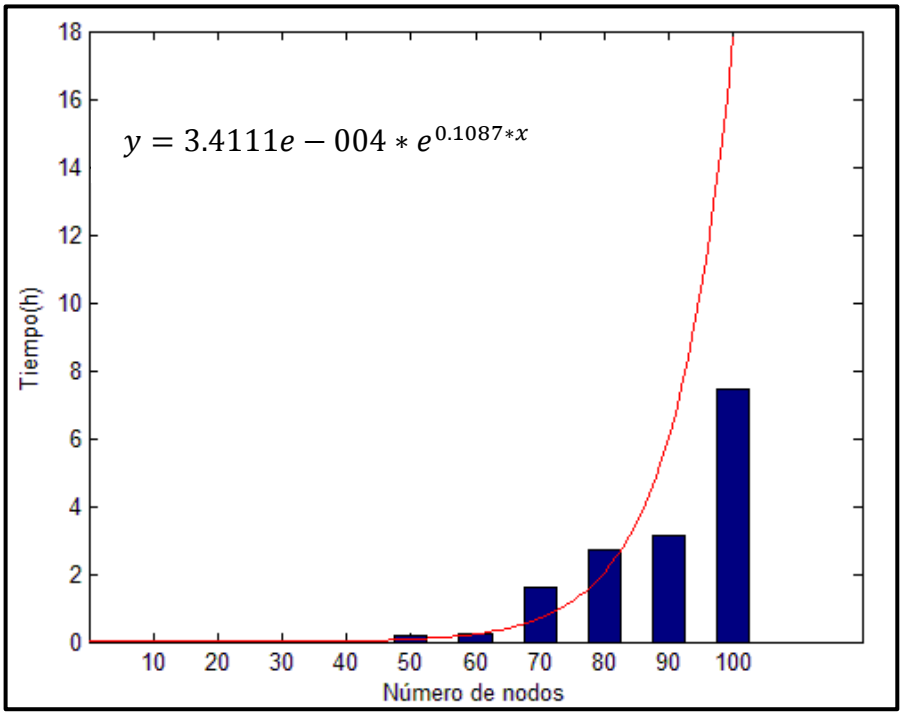


Fuente: Autores

La Figura 5 señala con mayor claridad, como es el comportamiento en el tiempo para las mismas condiciones descritas anteriormente, al igual que la regresión exponencial de los datos, que presentan una correlación del 0.9697. Se observa que para un número de 100 nodos, el algoritmo tarda en solucionarlo 7.46 horas. Aunque es evidente que la solución corresponde a una circunferencia, hacer que el ordenador entienda esto, solo es posible en la medida que el programa valide las mejores soluciones hasta encontrar la óptima, teniendo en cuenta que no se

está explorando en su totalidad el espacio de soluciones. Esto es un tiempo pequeño en comparación con el tiempo en que se tardaría explorar todas las soluciones. Tomando como ejemplo el tiempo en explorar tan solo 100 caminos, un algoritmo en MATLAB® tarda 0,0067[s], por lo tanto, para revisar una cantidad de $100!$ de soluciones, sería necesario emplear un tiempo de computo aproximado de $1.042 * 10^{152}$[s], que es un tiempo no razonable para aplicar.

Figura 5. Gráfica de Tiempo Vs Número de nodos, distribuidos en una circunferencia.



Fuente: Autores.

4.3.2 Validación con distribuciones de nodos del problema TSP

Para las pruebas de configuraciones de nodos del problema TSP, el algoritmo requiere de las coordenadas espaciales para cada una de las topologías. Además se debe asignar una condición de parada al algoritmo, en nuestro caso se tomó como condición el valor óptimo conocido en cada una de ellas.

4.3.2.1 Prueba con eil51

La configuración eil51 tiene un óptimo hallado hasta el momento de 426. La prueba se inicia ingresando sus coordenadas, y luego se ejecuta el algoritmo. En la Tabla 5 se presentan los resultados obtenidos para 5 pruebas independientes. Se puede deducir que al comparar el valor promedio obtenido con el mejor valor para esta configuración el error es 4.38%.

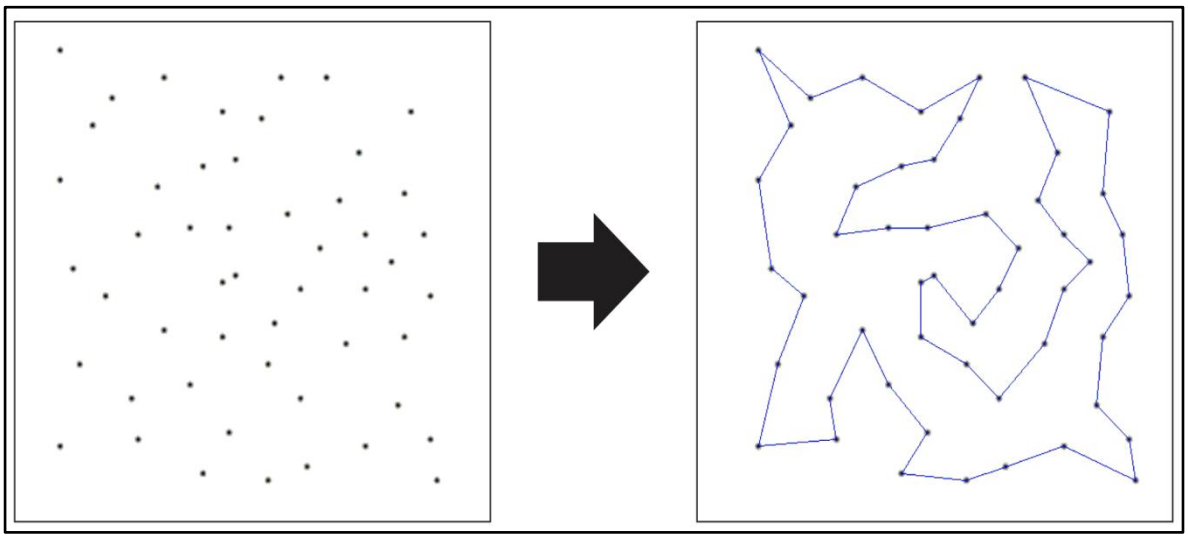
Tabla 5. Datos de longitud, número de iteraciones y tiempo para eil51.

Pruebas	1	2	3	4	5	Promedio
Longitud	444.61	447.05	448.19	445.9673	437.64	444.69
Iteraciones	59983	51562	34148	27230	93560	53296.6
Tiempo[s]	3154.8	2716.0	1816.8	1452.1	4991.8	2826.3

Fuente: Autores.

La Figura 6 corresponde al mejor recorrido encontrado para las pruebas realizadas. En la parte izquierda está la distribución de nodos mientras la parte derecha es la ruta óptima que debe seguir el robot móvil.

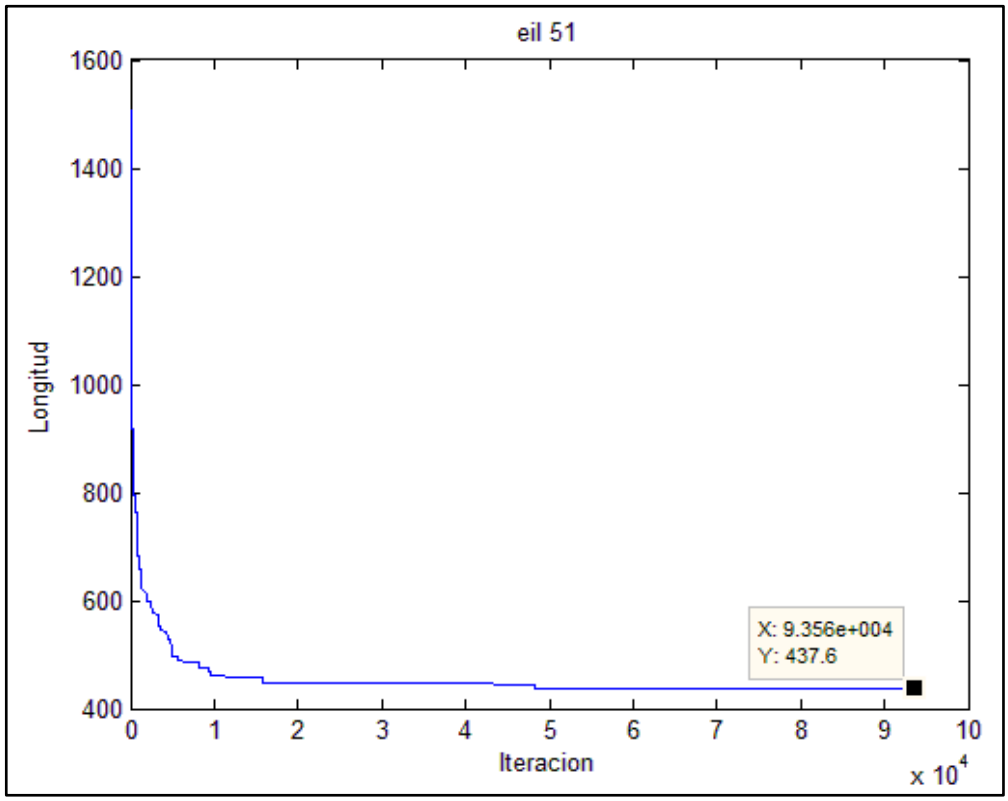
Figura 6. Mejor recorrido obtenido para la configuración eil51**.**



Fuente: Autores.

La Figura 7 es la variación de la longitud de la ruta, con respecto al número de ciclos ejecutados. En ella se puede apreciar que en las primeras 10000 iteraciones la gráfica presenta una caída rápida, comparada con los valores de las siguientes iteraciones.

Figura 7. Gráficas de longitud en función del número de iteraciones para eil51.



Fuente: Autores

4.3.2.2 Prueba con berlin52

En la Tabla 6, se presentan los resultados obtenidos, para 5 pruebas independientes, de la configuración berlin52. Para esta configuración el algoritmo presenta un error del 5.56%, contrastando el valor promedio obtenido con respecto al mejor recorrido conocido que es 7542.

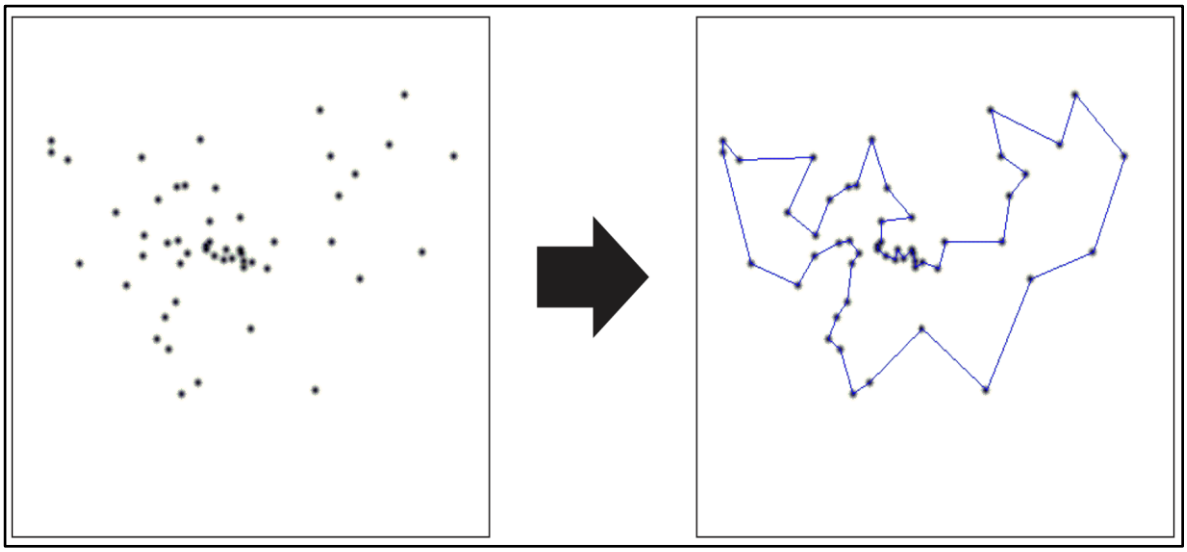
Tabla 6. Datos de longitud, número de iteraciones y tiempo para berlin52.

Pruebas	1	2	3	4	5	Promedio
Longitud	7973.3	7972.8	8072.8	8089.9	7700.1	7961.78
Iteraciones	98750	121241	21852	86934	69950	79745.4
Tiempo[s]	5339.7	6619.3	11619	4787.9	3753.0	6423.78

Fuente: Autores.

La Figura 8 presenta el mejor resultado encontrado para las pruebas realizadas para la distribución de nodos berlin52.

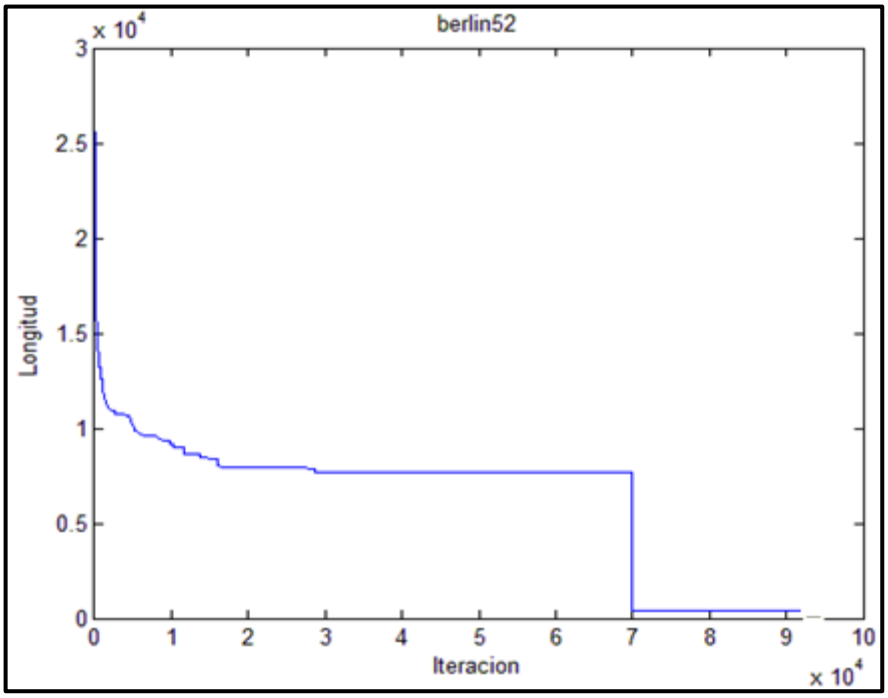
Figura 8. Mejor resultado obtenido para berlin52.



Fuente: Autores.

La Figura 9 es la variación de la longitud de la ruta con respecto al número de ciclos ejecutados, en la cual se puede apreciar que el algoritmo encuentra un valor óptimo alrededor de 30000 iteraciones, y converge alrededor de 70000 iteraciones.

Figura 9. Gráficas de longitud en función del número de iteraciones para berlin52.



Fuente: Autores.

4.3.2.3 Prueba con st70

La configuración st70 es una distribución de 70 nodos prestablecidos. Esta presenta un óptimo hallado hasta el momento de 675. Los resultados obtenidos para 5 pruebas del algoritmo se muestran en la Tabla 7. En esta se puede ver que la mejor ruta óptima encontrada es de valor 686.18, y el promedio para las 5 pruebas es de 702.86. Al comparar el mejor valor conocido de 675, con el valor promedio encontrado, se presentó un error del 4.13%.

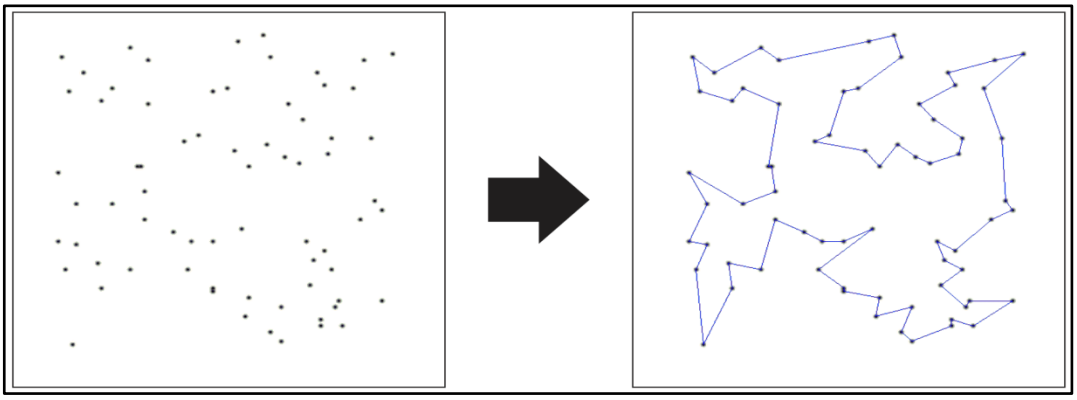
Tabla 7. Datos de longitud, número de iteraciones y tiempo para st70.

Pruebas	1	2	3	4	5	Promedio
Longitud	686.18	711.38	713.02	702.09	701.63	702.86
Iteraciones	60820	170765	87130	57199	46603	84503.4
Tiempo[s]	6092.3	17350	9043.3	5683.3	4685.3	8570.84

Fuente: Autores.

La Figura 10 corresponde al mejor resultado encontrado para pruebas realizadas para la configuración st70.

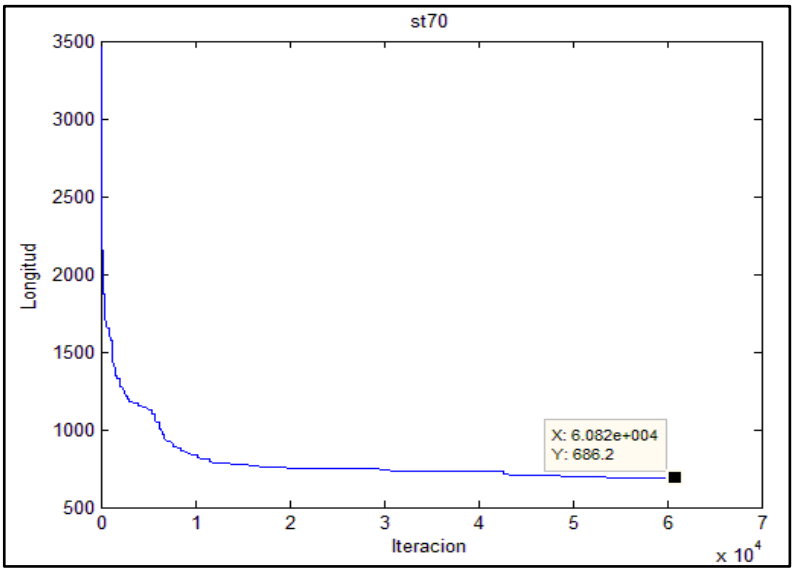
Figura 10. Mejor resultado obtenido para st70.



Fuente: Autores.

La Figura 11 es la gráfica de variación de la longitud de la ruta óptima con respecto al número de ciclos ejecutados; en esta se puede deducir que en aproximadamente 10000 iteraciones el algoritmo empieza a converger.

Figura 11. Gráficas de longitud en función del número de iteraciones para st70.



Fuente: Autores.

4.3.2.4 Prueba con eil76

En la Tabla 7 se presentan los resultados obtenidos para 5 pruebas independientes. Para este caso se presentó un error del 8.89%, al contrastar el valor óptimo encontrado con el mejor valor conocido que es 538.

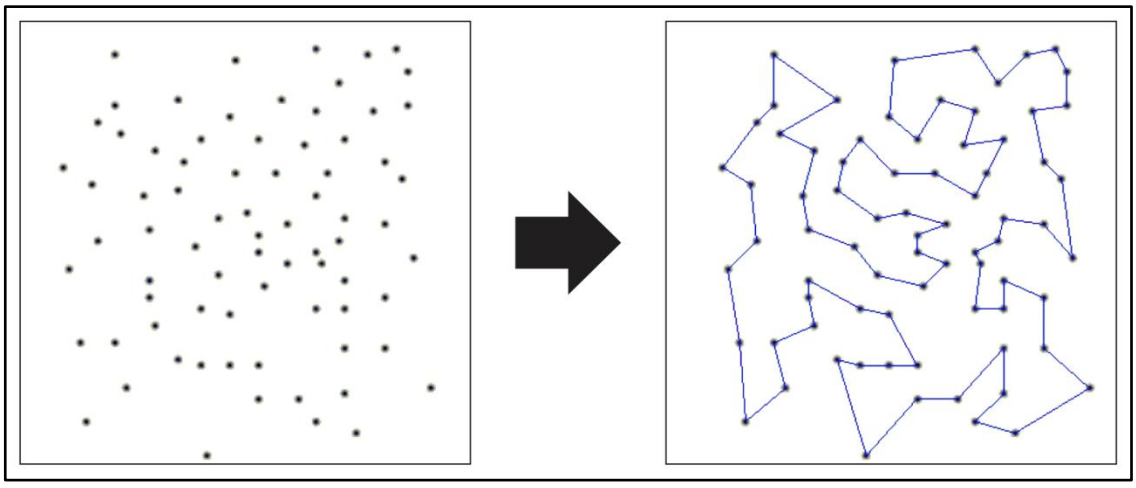
Tabla 8. Datos de longitud, número de iteraciones y tiempo para eil76.

Pruebas	1	2	3	4	5	Promedio
Longitud	580.55	581.87	592.05	589.01	585.72	585.84
Iteraciones	118193	146514	177535	108586	41811	118527.8
Tiempo[s]	14558	17769	21304	13439	5219.0	14457.8

Fuente: Autores.

La Figura 12 representa el mejor recorrido encontrado por el algoritmo, para las cinco pruebas realizadas, correspondiente a la configuración eil76.

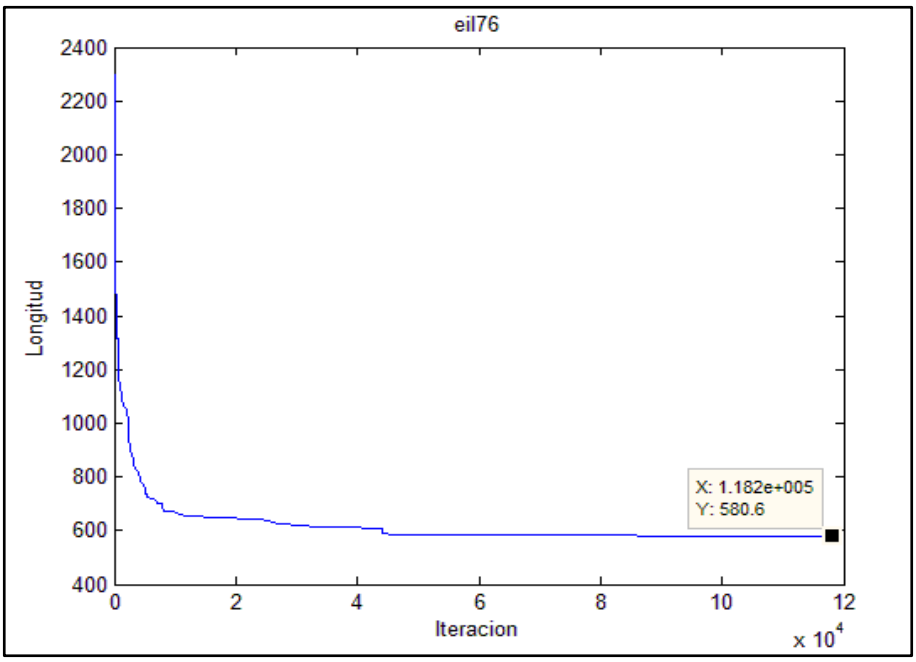
Figura 12. Mejor resultado obtenido para eil76.



Fuente: Autores.

La Figura 13 es la gráfica de variación de la longitud del recorrido, con respecto al número de ciclos ejecutados. En esta se puede ver que el algoritmo empieza a converger en aproximadamente 40000 iteraciones.

Figura 13. Gráficas de longitud en función del número de iteraciones para eil76.



Fuente: Autores.

4.3.2.5 Prueba con kroA100

En la Tabla 9 se muestran los resultados obtenidos para cuatro pruebas independientes. En este caso se presentó un error del 6.3% con respecto al mejor valor conocido que es de 21282.

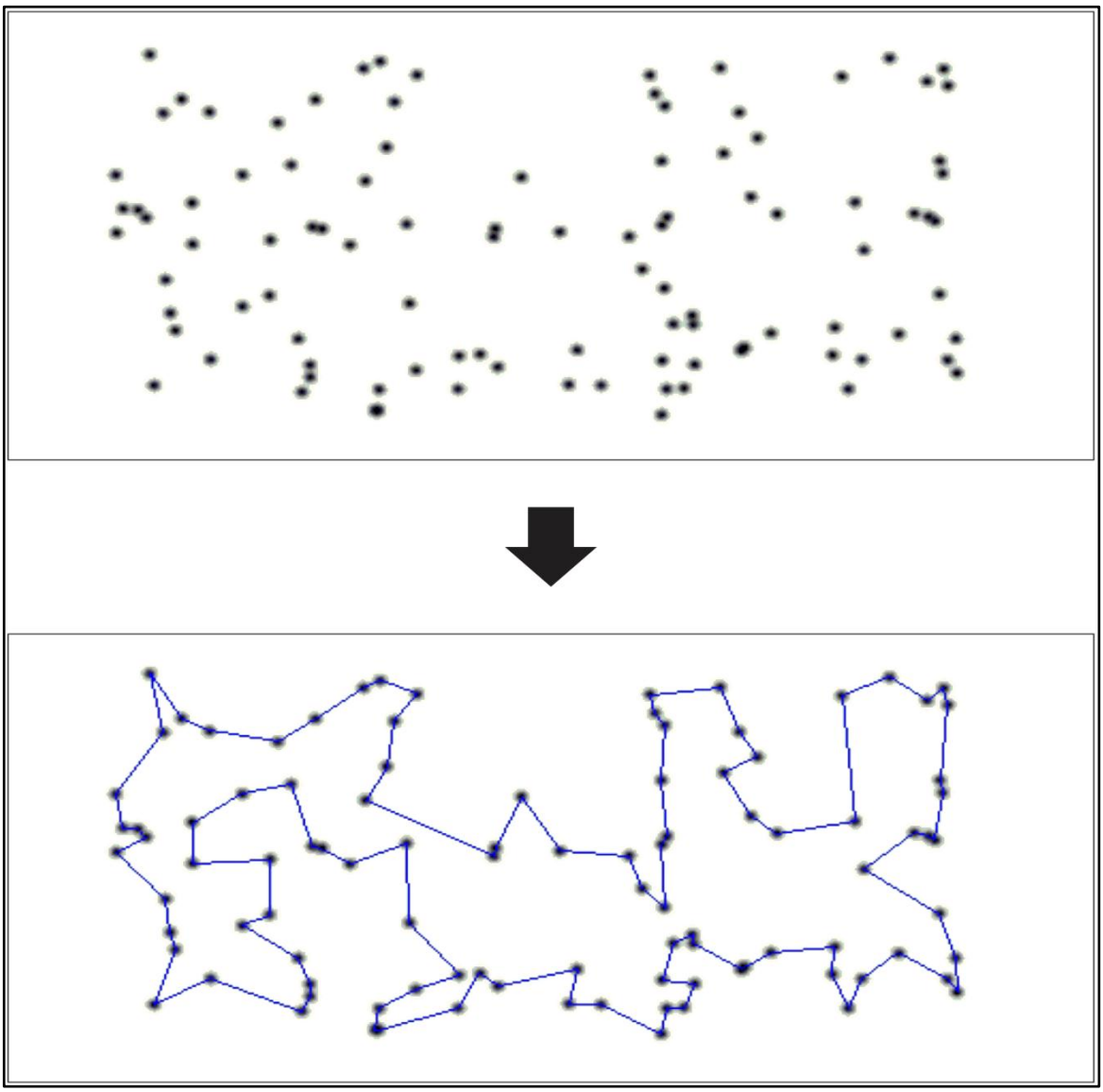
Tabla 9. Datos de longitud, número de iteraciones y tiempo para kroA100.

Pruebas	1	2	3	4	Promedio
Longitud	22287	22461	23619	22124	22622.75
Iteraciones	991324	950286	365193	1113814	855154.25
Tiempo[s]	225270	213310	84278	251790	193662

Fuente: Autores.

La Figura 14, corresponde al mejor recorrido encontrado por el algoritmo en las cuatro pruebas realizadas y para la configuración kroA100.

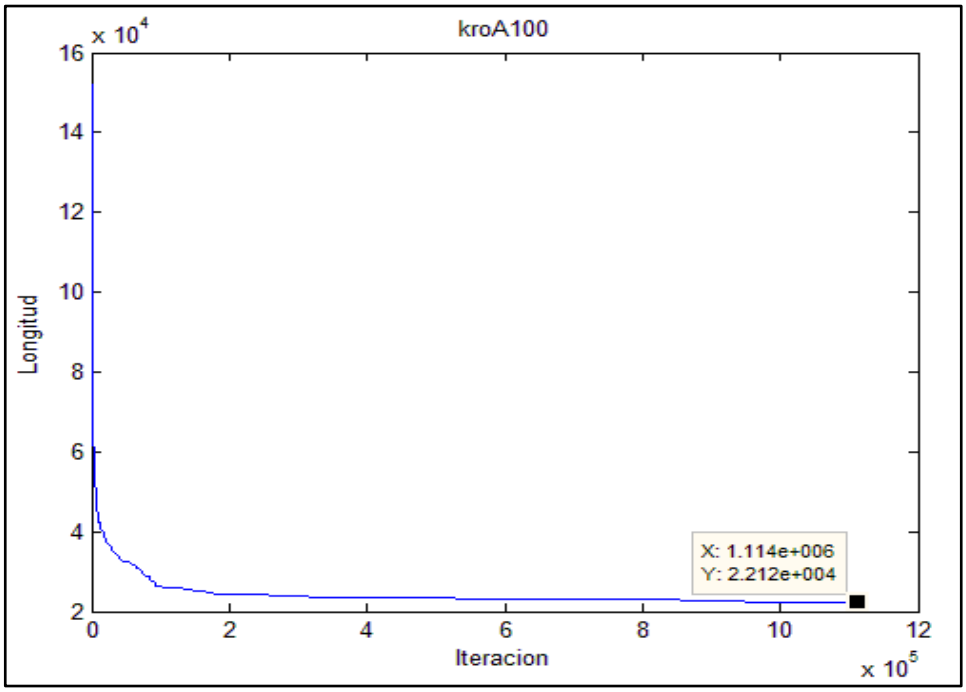
Figura 14. Mejor resultado obtenido para kroA100.



Fuente: Autores.

La Figura 15 corresponde con la característica de longitud de la ruta óptima, respecto al número de ciclos ejecutados, en la cual se puede observar que el algoritmo empieza a converger alrededor de 200000 iteraciones. De este valor en adelante, los cambios en longitud son relativamente pequeños comparados con el número de iteraciones.

Figura 15. Gráficas de longitud en función del número de iteraciones para kroA100.



Fuente: Autores.

La Tabla 10, es un resumen de los datos encontrados en las configuraciones anteriormente analizadas, adicionalmente se observa cual fue el espacio promedio de soluciones revisados del total de posibilidades.

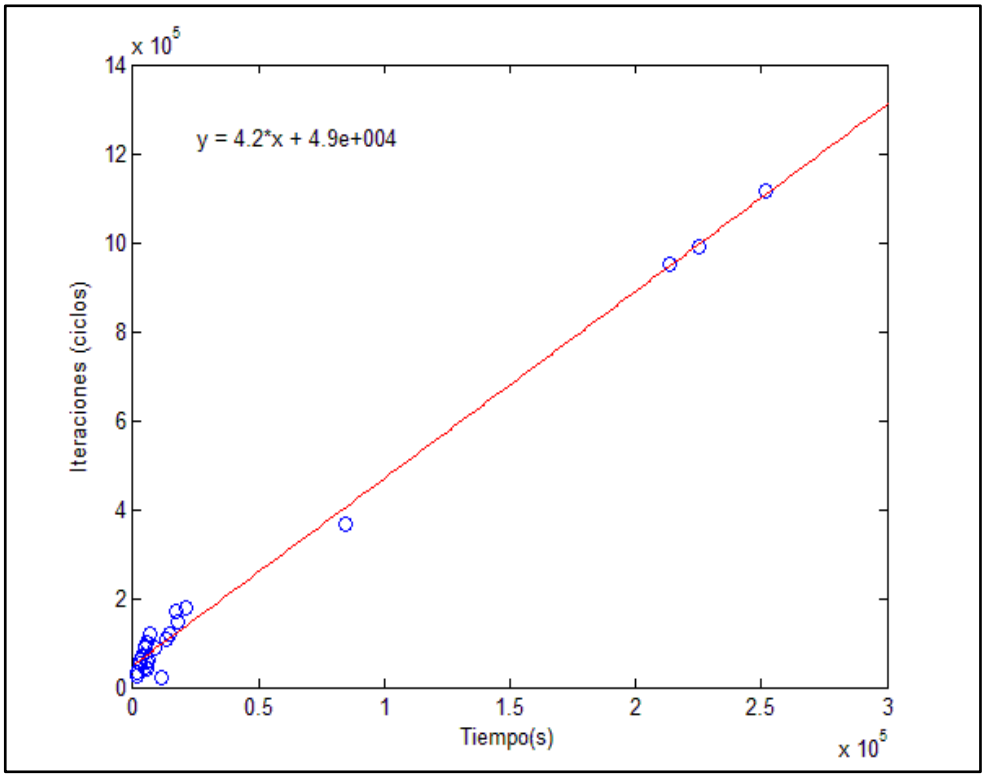
Tabla 10. Comparación del porcentaje del espacio de soluciones revisado, mejor valor conocido, valores óptimos encontrados, y error para las configuraciones analizadas anteriormente.

Configuración	% Espacio de soluciones revisado	Mejor valor conocido	Mejor óptimo encontrado	Valor promedio	Error[%]
eil51	1.7e-58	426	437.64	444.69	4.38
berlin52	5.1e-60	7542	7700.1	7961.78	5.56
st70	4.9e-92	675	686.18	702.86	4.13
Eil76	4.8e-103	538	580.55	585.84	8.89
kroA100	9.2e-149	21282	22124	22622.75	6.30

Fuente: Autores.

Para mostrar el comportamiento del algoritmo se realizó el ajuste a una regresión lineal para cada una de las configuraciones propuestas de la TSPLIB con el objetivo de generalizar el comportamiento del algoritmo respecto al tiempo. Los resultados se muestran en la Figura 16, donde se pueden observar cuatro puntos alejados tanto en tiempo como en cantidad de iteraciones, además se encontró un coeficiente de correlación de 0.9961, que permite definir la existencia de una relación fuerte entre los datos.

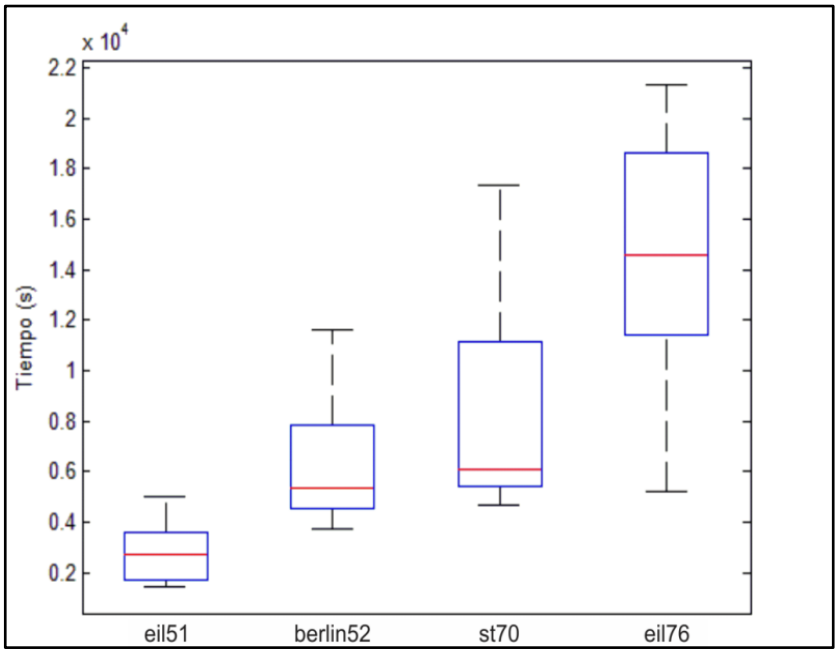
Figura 16. Curva de iteraciones vs tiempo.



Fuente: Autores.

La Figura 17 indica los valores promedio, para cada una de las configuraciones respecto al tiempo. Se observa cómo a medida que se incrementa el número de nodos, se incrementa el tiempo y el rango de sus intervalos.

Figura 17. Valores promedio respecto del tiempo para algunas de las configuraciones analizadas anteriormente.



Fuente: Autores.

Comparando los resultados obtenidos entre las configuraciones eil51, berlin52, st70, eil76 y kroA100 y los nodos distribuidos en el perímetro de una circunferencia, se encuentra que los tiempos varían considerablemente entre configuraciones simétricas y otras que no lo son. Por otra parte no se encontró el valor exacto en las configuraciones de la TSPLIB a diferencia con lo que ocurre con la circunferencia en el que no se presenta un error.

4.4 SEGUNDA ACTIVIDAD: SOLUCIÓN DE LABERINTOS

La segunda actividad propuesta corresponde a la solución de laberintos. Este es un caso que se puede ampliar a situaciones donde se requieren trayectorias con obstáculos. Para realizar esta propuesta de trabajo fue necesario realizar algunas modificaciones al algoritmo original.

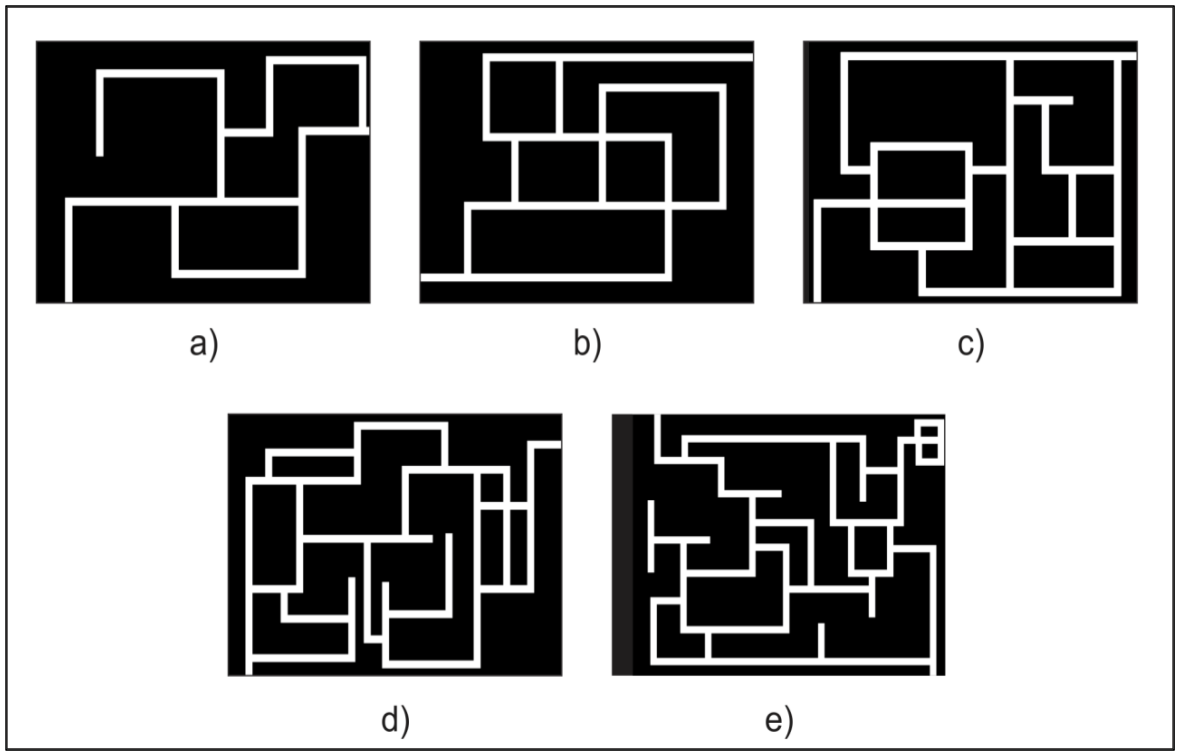
4.4.1 Modificaciones del algoritmo para la solución de laberintos

Para encontrar la trayectoria óptima en un laberinto, es necesario que se tenga conocimiento previo sobre el entorno de operación en el cual debe desenvolverse el robot móvil, esto con el fin de relacionar un grafo que describa dicho laberinto. Aunque el propósito del presente trabajo es implementar el algoritmo gotas de agua virtuales inteligentes, fue necesario desarrollar una programación independiente del algoritmo para poder describir el entorno de trabajo para las gotas de agua. Para ello se hizo un procesamiento digital de imágenes, realizando operaciones de erosión, dilatación y segmentación por áreas de la imagen que representa el laberinto.

Los laberintos seleccionados y utilizados para probar el algoritmo IWD se muestran en la Figura 18, los cuales están nombrados con las letras *a, b, c, d y e*. Estos laberintos presentan características que permiten una fácil descripción del grafo y en consecuencia un procesamiento de imágenes más sencillo a partir de su morfología. Estos laberintos prototipo, cuentan con las siguientes características: un ancho del camino de 32 pixeles, una separación entre caminos horizontales mayor a 90 pixeles y una separación entre caminos verticales mayor a 50 pixeles, con respecto a la calidad de la imagen, esta debe estar en escala de grises.

El algoritmo propuesto para la realización del grafo se basa en operaciones de morfología sobre la imagen adquirida del laberinto. Inicialmente la imagen se binariza con el objetivo de encontrar el esqueleto del laberinto, a partir de la erosión de la imagen binarizada, con un elemento estructurante cuadrado. Teniendo el esqueleto de la imagen se separan los elementos verticales y horizontales de la imagen; para esto se realiza una dilatación con un elemento estructurante tipo línea, tanto vertical como horizontal, dando como resultado dos máscaras, las cuales se multiplican para encontrar los vértices del laberinto.

Figura 18. Laberintos utilizados para comprobar el algoritmo IWD.

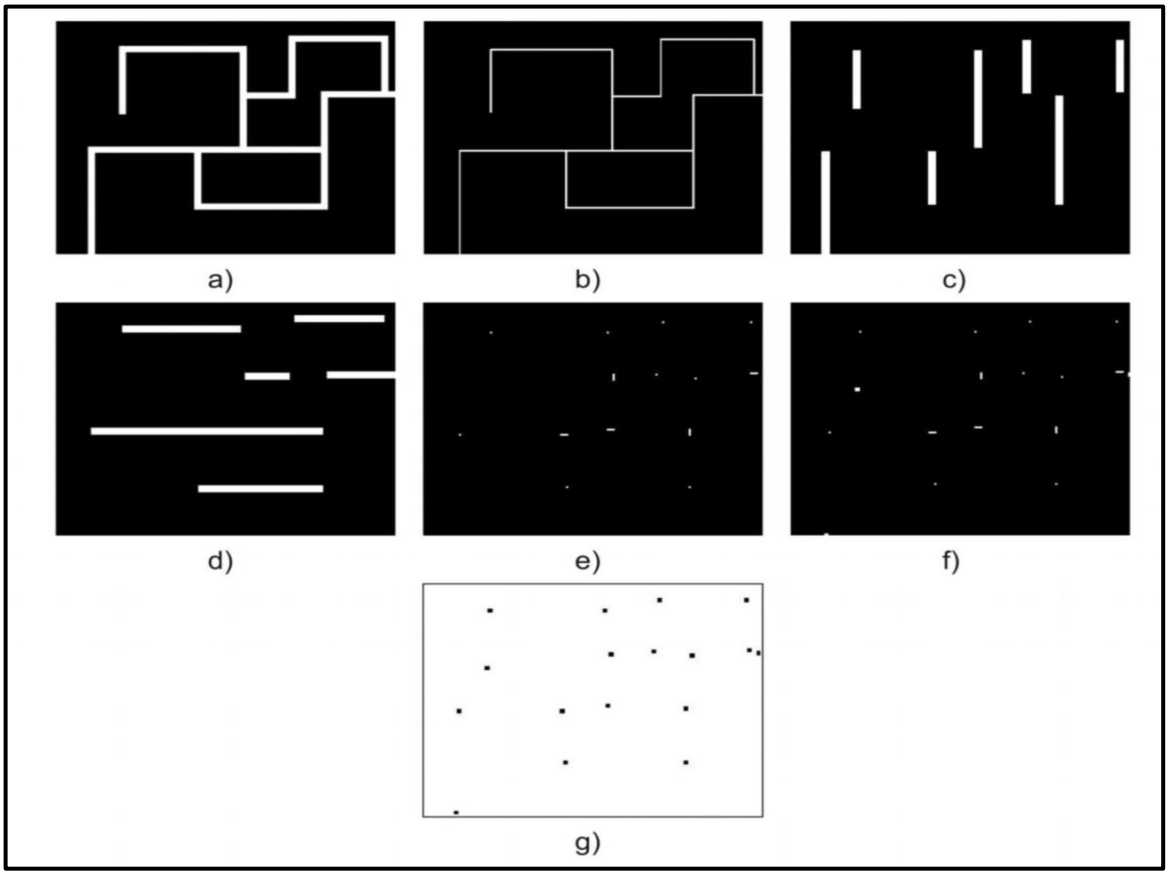


Fuente:[2]

A partir de los segmentos verticales y horizontales, se realiza una segmentación por áreas y se encuentran los puntos iniciales y finales de cada segmento. Multiplicando las intersecciones con esta nueva máscara se halla el grafo correspondiente del laberinto a tratar. Para una mejor visualización del grafo se encuentra el punto medio de cada nodo y se genera un cuadrado en cada uno de estos puntos medios. Finalmente se invierte la imagen para visualizar los nodos como puntos negros.

Los pasos para realizar esta descripción se pueden observar en la Figura 19, donde: a) es la imagen original, b) es el esqueleto de la imagen, c) es la dilatación vertical, d) es la dilatación horizontal, e) es la intersección, f) es el total de los vértices y g) generación de los puntos en los vértices.

Figura 19. Operaciones morfológicas para la obtención del grafo.



Fuente: Autores.

Se observa en la Figura 19, que es evidente que las gotas de agua virtuales inteligentes, ya no visitarán todos los nodos, sino exclusivamente los nodos que unan un punto inicial y un punto final, que serán definidos por el usuario. Por otra parte, las gotas de agua no inicializaran en nodos al azar, sino por el contrario, todas las gotas empezaran en un nodo inicial y terminan en un nodo final definido claramente.

Otra situación que se presenta es la manera de enlazar los nodos, ya que cada nodo no se une con todos los demás, por el contrario, estos presentan un número menor de enlaces. Por consiguiente, se hace necesario generar un mapeo de los vértices que enlazan los nodos del grafo del laberinto, teniendo entonces un

número menor de posibilidades de enlazar dichos nodos, a diferencia de la primera actividad, en donde todos los nodos tienen posibilidades de enlace. Para identificar los diferentes enlaces en el laberinto se utiliza una técnica de segmentación por áreas que describe el grafo tanto de sus líneas verticales como horizontales. Para posteriormente revisar en cada segmento los nodos del grafo que se conectan y finalmente actualizar el listado de nodos visitados para cada uno de los vértices.

4.4.2 Práctica experimental para la segunda actividad

Se propone validar el comportamiento del algoritmo, frente a la solución de laberintos, a medida que aumenta su dificultad. Para esto se ejecutaron 10 pruebas a los laberintos seleccionados para esta actividad; que son los correspondientes a la Figura 18.

En la Tabla 11 se resumen las características de estos laberintos prototipo, en donde se señala la cantidad de vértices, la longitud de la ruta óptima y el tiempo que tarda el programa diseñado para el mapeo del laberinto. Estos datos han sido tomados directamente de la morfología de los laberintos y de pruebas realizadas para cada una de los laberintos.

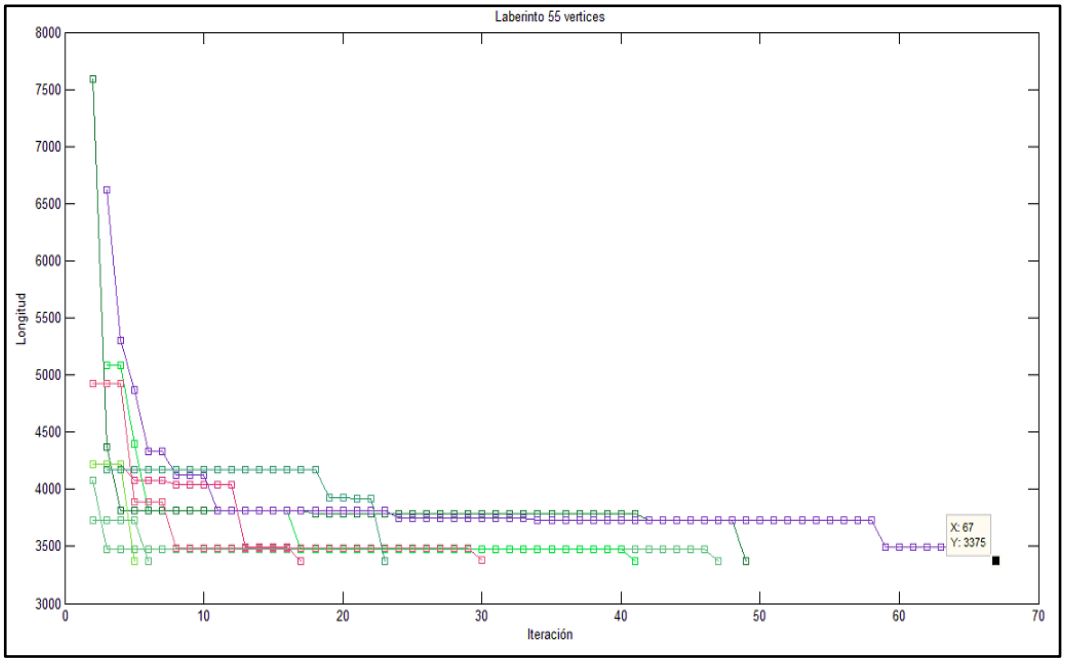
Tabla 11. Características de los laberintos prototipo utilizados.

Laberinto	Numero de vértices	Longitud ruta óptima	Tiempo(s) Mapeo del laberinto
a	17	1939	6.2
b	18	2922	9.2
c	29	3200	20.5
d	41	3673	39.8
e	55	3377	55.2

Fuente: Autores.

A manera de verificación, la Figura 20 muestra los resultados de las 10 pruebas realizadas al laberinto *e)*, en esta se observa la convergencia, para los 10 casos ejecutados. Estos resultados resaltan la calidad estocástica del algoritmo.

Figura 20. Resultados obtenidos de las 10 pruebas realizadas con el laberinto prototipo *e* de la Figura 18.



Fuente: Autores.

La Tabla 12 representa los resultados para los cinco laberintos propuestos, tanto del valor de iteración en que converge y el tiempo en segundos que tarda en llegar al valor óptimo, después de haber caracterizado el mapa del laberinto.

Tabla 12. Resultados de iteraciones para los cinco laberintos propuestos.

Laberinto	Datos de iteración										Media	Desviación
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10		
a	4	4	2	1	4	3	2	3	2	4	2.7	1.06
b	35	114	29	101	144	99	29	26	434	35	123	126
c	59	280	63	77	31	209	354	24	99	67	126.3	114.1
d	174	52	110	86	126	173	26	56	11	219	103.3	69.5
e	91	170	142	103	9	95	189	103	264	9	117.5	78.3

Fuente: Autores.

La Tabla 13 presenta los datos de tiempo, que tarda el algoritmo en llegar al valor óptimo.

Tabla 13. Resultados de tiempo para llegar al valor óptimo.

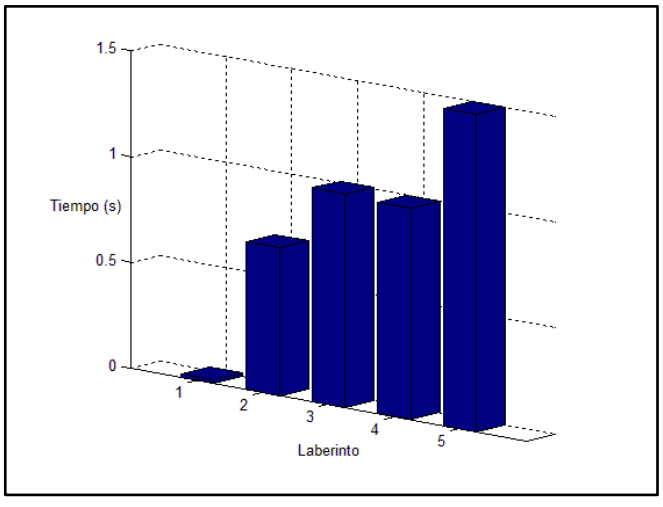
Laberinto	Datos de tiempo [s]										Media	Desviación
	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10		
a	0,02	0,02	0,01	0,01	0,03	0,02	0,03	0,02	0,01	0,02	0.017	0.006
b	0,2	0,6	0,1	0,5	0,8	0,5	0,1	0,1	2,5	1,2	0.7	0.7
c	0,5	2,4	0,7	0,6	0,2	1,6	2,9	0,2	0,7	0,5	1.01	0.93
d	1,8	0,5	1,4	0,9	1,1	1,4	0,2	0,5	0,1	1,8	1	0.6
e	1,1	2,1	1,7	1,2	0,1	1,2	2,4	1,5	3,4	0,1	1.5	1

Fuente: Autores.

El diagrama de barras mostrado en la Figura 21 señala con más claridad el aumento de tiempo en la medida que se incrementa el número de vértices del laberinto. Los resultados presentan una alta dispersión, y esto se debe al reducido número de posibilidades o caminos en cada intersección.

El mayor tiempo promedio es de 1.5 [s], encontrado en el laberinto *e)* que cuenta con 55 vértices; el menor tiempo promedio es de 0.017[s], que se da en el laberinto *a*, con 17 vértices.

Figura 21. Gráfica de tiempo en función al número de vértices de un laberinto.



Fuente: Autores.

4.5 INTERFAZ GRÁFICA PARA USUARIO

La interfaz gráfica para usuario del algoritmo gotas de agua virtuales inteligentes, se desarrolló en GUIDE de MATLAB®. Una foto de la interfaz se muestra en la Figura 22 donde se observa el menú en la parte inferior derecha; este tiene cinco configuraciones de nodos y la del laberinto. En la parte inferior izquierda se encuentran los botones de iniciar y parar; también hay dos recuadros para ingresar el número de iteraciones y el número de gotas, además de esto se muestra el valor óptimo conocido para la topología seleccionada. En la parte superior aparecen la gráficas de la configuración seleccionada, y de longitud vs iteraciones para el mejor recorrido encontrado por el algoritmo.

Pasos para ejecutar las configuraciones de nodos en la interfaz

1. Seleccionar la configuración deseada
2. Ingresar el número de iteraciones y oprimir el botón iniciar
3. Después de esperar un tiempo adecuado se oprime el botón parar (o cuando termine de iterar), se mostrarán las gráficas correspondientes y el valor de longitud encontrado por el algoritmo.

Figura 22. Interfaz gráfica de usuario del algoritmo gotas de agua virtuales inteligentes, programada en MATLAB®

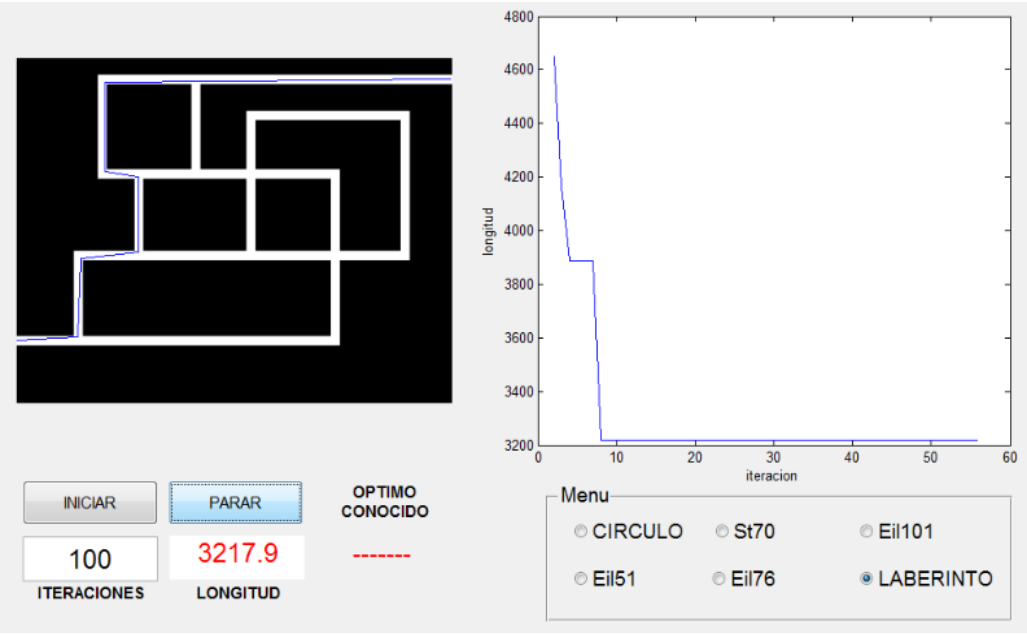
Fuente: Autores.

Los pasos para ejecutar la opción de laberinto son los siguientes:

1. Seleccionar la opción laberinto
2. Aparecerá una ventana con algunos laberintos, el usuario selecciona el que desee
3. Ingresar el número de gotas, e iniciar
4. Seleccionar con el mouse los puntos inicial y final del laberinto, para que el algoritmo encuentre la ruta óptima que necesitaría el robot para desplazarse.

La Figura 23 corresponde a un ejemplo de solución de un laberinto. En esta se puede apreciar el funcionamiento del algoritmo.

Figura 23. Interfaz gráfica de usuario programada en MATLAB® (ejemplo de laberinto).



Fuente: Autores.

5. CONCLUSIONES

- Se cumplieron todos los objetivos propuestos para el presente trabajo de grado en la modalidad de investigación.

- El algoritmo gotas de agua virtuales inteligentes es aplicable en la planeación de rutas para robots móviles, por lo que se cumplieron los objetivos propuestos para su aplicación en ingeniería electrónica.

- La simetría en las rutas reduce el tiempo de cómputo para la estimación de la ruta óptima, debido a que el listado de nodos visitados se trasforma en una matriz simétrica, reduciendo así el número de posibilidades por nodo de los enlaces a seleccionar.

- El aumento de número de nodos incrementa significativamente el tiempo de cómputo, como se puede verificar en la Tabla 1, y las gráficas correspondientes a las Figura 4, y Figura 5.

- El aumento del número de gotas de agua virtuales inteligentes generadas en el algoritmo para la solución de laberintos, reduce el número de iteraciones para encontrar la solución; esto se debe a que se produce un aumento en el número de posibilidades, es decir, un aumento en el número de soluciones probadas para cada iteración.

- Los casos de la TSP y solución de laberintos son ejemplos que pueden generalizarse para la solución de planeación de rutas con obstáculos.

- El procesamiento de imágenes desarrollado para la descripción del grafo del laberinto, es una programación sencilla, que solo es aplicable a

laberintos similares a las características que se describen en el documento. Para el caso de laberintos más complejos se requiere de un procesamiento de imágenes diferente al tratado, que deben tener en cuenta el ancho del camino, el aumento del número vértices, el tipo de laberinto (circular o recto), entre otros.

6. RECOMENDACIONES Y TRABAJO FUTURO

Este documento puede tomarse como base para un trabajo posterior, en el cual el algoritmo se programe para su operación paralela, es decir, que el software pueda coger inicialmente todas las gotas, y no solo una como se desarrolló en este trabajo. También queda la posibilidad de implementarlo en un lenguaje de bajo nivel, tal que permita mejorar la velocidad de cómputo del algoritmo, de tal manera que se programe en una FPGA o un microcontrolador o un sistema similar.

7. BIBLIOGRAFÍA

[1] García G. “Introducción a la Resolución de Problemas con Algoritmos Genéticos,” pp. 2,3,6,12, 2010. Disponible en: https://www.google.com.co/webhp?hl=es&tab=ww#hl=es&output=search&sclient=psyab&q=introducci%c3%93n+a+la+resoluci%c3%93n+de+problemas. [Consultado el 21 de agosto de 2012].

[2] Garro Licón B. A. “Estudio comparativo de diferentes técnicas bio-inspiradas para encontrar el camino más corto de un robot móvil dentro de un laberinto,” pp. 52, 2007. Disponible en la página web: http://www.cic.ipn.mx/posgrados/images/sources/cic/tesis/B081504.pdf. [Consultado el 23 de abril de 2013].

[3] González Velarde, J. L y Ríos Mercado, R. “Investigación de operaciones en acción,” *Ingenierías*, vol. II, no. 4, pp. 18–23, 2000. Disponible en web: http://ingenierias.uanl.mx/9/pdf/9_Roger_Rios_et_al_Investigacion_de_oper.pdf. [Consultado el 3 de abril de 2013].

[4] Greco, Federico. "*Traveling Salesman Problem"*. pp 1-2, 10-14, 28-31. InTech, 2008. Disponible en la página web: http://www.intechopen.com/books/traveling_salesman_problem. [Consultado el 15 de julio de 2012].

[5] Haibin, D; Senq, Il y Wu, J. “Novel intelligent water drops optimization approach to single UCAV smooth trajectory planning,” *Aerospace Science and Technology*, vol. 13, no. 8, pp. 442–449, Dec. 2009. Disponible en: http://linkinghub.elsevier.com/retrieve/pii/S1270963809000406. [Consultado el 23 de junio de 2012].

[6] Kamkar, I; Akbarzadeh-T, M.-R y Yaghoobi, M. “Intelligent water drops a new optimization algorithm for solving the Vehicle Routing Problem,” *2010 IEEE International Conference on Systems, Man and Cybernetics*, pp. 4142–4146, Oct. 2010. Disponible en la página web: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5642405. [Consultado el 24 de julio de 2012].

[7] Kesavamoorthy, R. “Solving Traveling Salesman Problem by Modified Intelligent Water Drop Algorithm,”. *Proceedings published by International Journal of Computer Applications® (IJCA)*, pp. 18-23, 2011. Disponible en: http://www.ijcaonline.org/icett/number2/icett016.pdf. [Consultado el 29 de marzo de 2013].

[8] Minetti, G. “Una Solución de Computación Evolutiva para el TSP, su Posible Aplicación en las Organizaciones,”. Tesis de maestría, vol. 702, pp. 84–92, 2000. Disponible en: http://sedici.unlp.edu.ar/bitstream/handle/10915/4059/Documento_completo.pdf?sequence=13.

[9] Ortiz Rodríguez, C. “Algoritmos heurísticos y metaheurísticos para el problema de localización de regeneradores.,” pp. 11–26, 2010. Trabajo de grado. Disponible en: http://eciencia.urjc.es/handle/10115/4129. [Consultado el 27 de marzo de 2013].

[10] Rayapudi, S. R. “An Intelligent Water Drop Algorithm for Solving Economic Load Dispatch Problem,”. World Academy of Science, Engineering and Technology 58 2011, *Department of Electrical and Electronics Engineering, J. N. T. University Kakinada, INDIA*, pp. 923–929, 2011. Disponible en: http://ieeexplore.ieee.org. [Consultado el 17 de julio de 2012].

[11] Shah-Hosseini, H. “Optimization with the Nature-Inspired Intelligent Water Drops Algorithm,”. *Proceedings of IEEE Congress on Evolutionary*

Computation, pp. 3226–3231, 2007. Department, Shahid Beheshti university, Teheran, Iran. Disponible en: http://ieeexplore.ieee.org. [Consultado el 24 de Julio de 2012].

[12] Shah-hosseini, H. "Optimization with the Nature-Inspired Intelligent Water Drops Algorithm,". *Faculty of Electrical and Computer Engineering, Shahid Beheshti University, G.C,* no. October, 2009. I-Tech, Viena, Austria. Disponible en: http://ieeexplore.ieee.org/Xplore/home.jsp. [Consultado el 1 de julio de 2012].

[13] Shah-Hosseini, H. "The intelligent water drops algorithm: a nature-inspired swarm-based optimization algorithm,". *International Journal of Bio-Inspired Computation*, vol. 1, no. 1/2, p. 71, 2009. Disponible en: http://inderscience.metapress.com/index/A4065612210T6130.pdf. [Consultado el 17 de julio de 2012].

[14] Shah-Hosseini, H. "An approach to continuous optimization by the Intelligent Water Drops algorithm,". *Procedia - Social and Behavioral Sciences*, vol. 32, no. 2010, pp. 224–229, 2012. Disponible en la página web: http://www.sciencedirect.com/science/journal/18770428/32. [Consultado el 11 de agosto de 2012].

[15] "TSPLIB." [En linea]. Disponible en la página web: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. [Consultado el 17 de abril de 2013].

[16] Yepes Piqueras, V. "Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW," pp. 22–34, 2002. Tesis Doctoral de la Universidad Politécnica de Valencia. Disponible en: http://riunet.upv.es/bitstream/handle/10251/2664/tesisUPV1497.pdf. [Consultado el 10 de agosto de 2012].