UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica

Trabajo de graduación presentado por Gabriela Iriarte Colmenares para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Aprendizaje Automático, Computación Evolutiva e Inteligencia de Enjambre para Aplicaciones de Robótica

Trabajo de graduación presentado por Gabriela Iriarte Colmenares para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2020

Vo.Bo.:

(f) _____

Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) _____

Dr. Luis Alberto Rivera Estrada

(f) _____

MSc. Carlos Esquit

(f) _____

Msc. Miguel Enrique Zea Arenales

Fecha de aprobación: Guatemala, 5 de diciembre de 2020.

Prefacio

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vitae eleifend ipsum, ut mattis nunc. Pellentesque ac hendrerit lacus. Cras sollicitudin eget sem nec luctus. Vivamus aliquet lorem id elit venenatis pellentesque. Nam id orci iaculis, rutrum ipsum vel, porttitor magna. Etiam molestie vel elit sed suscipit. Proin dui risus, scelerisque porttitor cursus ac, tempor eget turpis. Aliquam ultricies congue ligula ac ornare. Duis id purus eu ex pharetra feugiat. Vivamus ac orci arcu. Nulla id diam quis erat rhoncus hendrerit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed vulputate, metus vel efficitur fringilla, orci ex ultricies augue, sit amet rhoncus ex purus ut massa. Nam pharetra ipsum consequat est blandit, sed commodo nunc scelerisque. Maecenas ut suscipit libero. Sed vel euismod tellus.

Proin elit tellus, finibus et metus et, vestibulum ullamcorper est. Nulla viverra nisl id libero sodales, a porttitor est congue. Maecenas semper, felis ut rhoncus cursus, leo magna convallis ligula, at vehicula neque quam at ipsum. Integer commodo mattis eros sit amet tristique. Cras eu maximus arcu. Morbi condimentum dignissim enim non hendrerit. Sed molestie erat sit amet porttitor sagittis. Maecenas porttitor tincidunt erat, ac lacinia lacus sodales faucibus. Integer nec laoreet massa. Proin a arcu lorem. Donec at tincidunt arcu, et sodales neque. Morbi rhoncus, ligula porta lobortis faucibus, magna diam aliquet felis, nec ultrices metus turpis et libero. Integer efficitur erat dolor, quis iaculis metus dignissim eu.

Índice

Lista de figuras

Lista de cuadros

Resumen

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras vitae eleifend ipsum, ut mattis nunc. Pellentesque ac hendrerit lacus. Cras sollicitudin eget sem nec luctus. Vivamus aliquet lorem id elit venenatis pellentesque. Nam id orci iaculis, rutrum ipsum vel, porttitor magna. Etiam molestie vel elit sed suscipit. Proin dui risus, scelerisque porttitor cursus ac, tempor eget turpis. Aliquam ultricies congue ligula ac ornare. Duis id purus eu ex pharetra feugiat. Vivamus ac orci arcu. Nulla id diam quis erat rhoncus hendrerit. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Sed vulputate, metus vel efficitur fringilla, orci ex ultricies augue, sit amet rhoncus ex purus ut massa. Nam pharetra ipsum consequat est blandit, sed commodo nunc scelerisque. Maecenas ut suscipit libero. Sed vel euismod tellus.

Proin elit tellus, finibus et metus et, vestibulum ullamcorper est. Nulla viverra nisl id libero sodales, a porttitor est congue. Maecenas semper, felis ut rhoncus cursus, leo magna convallis ligula, at vehicula neque quam at ipsum. Integer commodo mattis eros sit amet tristique. Cras eu maximus arcu. Morbi condimentum dignissim enim non hendrerit. Sed molestie erat sit amet porttitor sagittis. Maecenas porttitor tincidunt erat, ac lacinia lacus sodales faucibus. Integer nec laoreet massa. Proin a arcu lorem. Donec at tincidunt arcu, et sodales neque. Morbi rhoncus, ligula porta lobortis faucibus, magna diam aliquet felis, nec ultrices metus turpis et libero. Integer efficitur erat dolor, quis iaculis metus dignissim eu.

Abstract

This is an abstract of the study developed under the

CAPÍTULO 1

Introducción

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque eget consequat risus. Praesent a quam lacinia, consequat eros id, auctor tellus. Phasellus a dapibus arcu, vitae luctus leo. Aliquam erat volutpat. Suspendisse ac velit quam. Nullam risus nibh, lobortis vehicula elit non, pellentesque volutpat odio. Donec feugiat porta sapien gravida interdum. Cras odio nunc, lobortis sed pellentesque imperdiet, facilisis eu quam. Praesent pharetra, orci at tincidunt lacinia, neque nulla ornare lacus, ut malesuada elit risus non mi. Fusce pellentesque vitae sapien sed mollis. Curabitur viverra at nulla vitae porta. In et mauris lorem.

Vestibulum faucibus fringilla justo, eget facilisis elit convallis sit amet. Morbi nisi metus, hendrerit quis pellentesque non, faucibus at leo. Proin consectetur, est vel facilisis facilisis, arcu felis vestibulum quam, et fringilla metus neque at enim. Nunc justo mauris, egestas quis maximus eget, viverra vehicula nunc. Fusce eu nulla elementum, condimentum diam at, aliquam leo. Nullam sed sodales enim, eu imperdiet risus. Aliquam ornare augue leo, fringilla mattis nunc facilisis eget. Nam faucibus, libero a aliquet fermentum, magna arcu ultrices lacus, a placerat tortor turpis ut purus.

Integer eget ligula non metus egestas rutrum sit amet ut tellus. Aliquam vel convallis est, eu sodales leo. Proin consequat nisi at nunc malesuada gravida. Aliquam erat volutpat. Aliquam finibus interdum dignissim. Etiam feugiat hendrerit nisl, hendrerit feugiat ex malesuada in. Cras tempus eget arcu vitae congue. Ut non tristique mauris. Vivamus in mattis ipsum. Cras bibendum, enim bibendum commodo accumsan, ligula nulla porttitor ex, et pharetra eros nisl eget ex. Morbi at semper arcu. Curabitur massa sem, maximus id metus ut, molestie tempus quam. Vivamus dictum nunc vitae elit malesuada convallis. Donec ac semper turpis, non scelerisque justo. In congue risus id vulputate gravida. Nam ut mattis sapien.

CAPÍTULO 2

Antecedentes

2.1. Megaproyecto Robotat - Fase II

En la segunda fase del proyecto Robotat [1] se elaboró un algoritmo basado en el Particle Swarm Optimization (PSO) para que distintos agentes llegaran a una meta definida (el mínimo de una función de costo). El algoritmo modificado elaboraba una trayectoria a partir de la actualización de los puntos de referencia a seguir, generados por el PSO clásico. Asimismo, se realizó distintas pruebas para encontrar los mejores parámetros del algoritmo PSO, probando distintas funciones de costo. Para realizar dicho algoritmo se tomó en cuenta que la velocidad de los robots diferenciales tiene un límite, además de tener que seguir las restricciones físicas de un robot real. Para simular estas restricciones se utilizó el software de código abierto Webots luego de haberlo implementado en Matlab como partículas sin masa ni restricciones.

En Webots también se implementó distintos controladores para que la trayectoria fuera suave y lo más uniforme posible. Entre los controladores que se implementaron están: Transformación de uniciclo (TUC), Transformación de uniciclo con PID (TUCPID), Controlador simple de pose (SPC), Controlador de pose Lyapunov-estable (LSPC), Controlador de direccionamiento de lazo cerrado (CLSC), Transformación de uniciclo con LQR (TUC-LQR), y Transformación de uniciclo con LQI (TUC-LQI). El controlador con el mejor resultado en esta fase fue el TUC-LQI.

2.2. Robotarium de Georgia Tech

El Robotarium del Tecnológico de Georgia en Estados Unidos desarrolló una mesa de pruebas para robótica de enjambre para que personas de todo el mundo pudieran hacer pruebas con sus robots. De este modo, ellos están apoyando a que más personas, sin importar sus recursos económicos, puedan aportar a la investigación de robótica de enjambre. Además, eso ayudaría a los investigadores a dejar de simular y probar sus algoritmos en prototipos

reales. Para utilizar la plataforma hay que descargar el simulador del Robotarium de Matlab o Python, registrarse en la página del Robotarium y esperar a ser aprobado para crear el experimento [2].

2.3. Wyss Institute Swarm Robots

El Instituto Wyss de Harvard desarrolló robots de bajo costo miniatura llamados *Kilobots* para probar algoritmos de inteligencia computacional de enjambre. Este tipo de robots es desarrollado para potencialmente realizar tareas complejas en el ambiente como lo son polinizar o creación de construcciones. Estos Kilobots cuentan con sensores, micro actuadores y controladores robustos para permitir a los robots adaptarse a los ambientes dinámicos y cambiantes [3].

2.4. Aplicaciones de Ant Colony Optimization (ACO)

En [4] puede encontrarse un trabajo similar, donde se aplica el método de ACO en un robot autónomo en una cuadrícula con un obstáculo estático. El robot lo colocan en la esquina inferior izquierda y se pretende que alcance el objetivo en la esquina superior derecha de la cuadrícula. La simulación de este trabajó se realizó en Matlab y se implementó una cuadrícula de 100x100 con la unidad como el tamaño de las aristas. Además, en la cuadrícula los agentes podían moverse norte, sur, este, oeste y en forma diagonal.

También en [5] se realizó una comparación de los algoritmos PSO y ACO, en donde se resaltó algunas ventajas y desventajas de cada uno de los métodos. El PSO es simple pero sufre al quedarse atascado con mínimos locales por la regulación de velocidad. Por otro lado, el ACO se adapta muy bien en ambientes dinámicos, pero el tiempo de convergencia puede llegar a ser muy largo (aunque la convergencia sí está garantizada).

CAPÍTULO 3

Justificación

En la segunda fase del proyecto Robotat de la Universidad del Valle de Guatemala se desarrolló un algoritmo basado en el algoritmo Particle Swarm Optimization para planificar una trayectoria óptima para que los robots llegaran a una meta determinada. También se elaboró el diseño de distintos controladores para asegurar que los robots recibieran velocidades coherentes con respecto a sus limitantes físicas. Ya que este algoritmo y controlador fueron exitosos en simulación, se desearía comparar con otros algoritmos para que, de este modo, el mejor algoritmo sea finalmente implementado en los robots físicos.

Como propuesta de algoritmo se tiene el Ant Colony Optimization(ACO), pues es otro de los más utilizados de la rama de inteligencia computacional de enjambre. Se debe de encontrar los mejores parámetros para que el algoritmo converja en un tiempo similar al logrado con el PSO de la fase II del proyecto Robotat. Asimismo se busca implementar los mismos controladores que en el proyecto mencionado anteriormente para que la comparación sea equitativa.

CAPÍTULO 4

Objetivos

4.1. Objetivo General

Implementar y verificar algoritmos de inteligencia de enjambre y computación evolutiva como alternativa al método de Particle Swarm Optimization para los Bitbots de UVG.

4.2. Objetivos Específicos

- Implementar el algoritmo Ant Colony Optimization (ACO) y encontrar el valor de los parámetros de las ecuaciones del ACO que permitan a los elementos de la colonia encontrar una meta específica, en un tiempo finito.
- Validar los parámetros encontrados por medio de simulaciones computarizadas que permitan la visualización del comportamiento de la colonia.
- Adaptar los modelos del movimiento y de la cinemática de los Bitbots, desarrollados en la fase anterior del proyecto, al algoritmo ACO.
- Validar el algoritmo ACO adaptado implementándolo en robots físicos simulados en el software WeBots, y comparar su desempeño con el del Modified Particle Swarm Optimization (MPSO).

CAPÍTULO 5

Alcance

Podemos usar Latex para escribir de forma ordenada una fórmula matemática.

CAPÍTULO 6

Marco teórico

6.1. Inteligencia Computacional Swarm

El campo de la inteligencia computacional Swarm parte de la idea de que individuos interactúan entre ellos para resolver un objetivo global por medio de intercambio de de su información local. De esta manera, la información se propaga más rápido y por tanto, el problema se resuelve de una manera más eficiente. El término *Swarm Intelligence* (SI) se refiere a esta estrategia de resolución de problemas colectiva, mientras que *Computational Swarm Intelligence* (CSI) se refiere a algorimos que modelan este comportamiento [6].

Al tener diversos estudios sobre distintos tipos de animales es posible crear algoritmos que modelen las situaciones. Algunos sistemas biológicos que han inspirado este tipo de algoritmos son: hormigas, termitas, abejas, arañas, escuelas de peces y bandadas de pájaros. Dos de los algoritmos que se estudiarán más adelante están basados en pájaros y hormigas, modelando su comportamiento individual y colectivo.

6.2. Particle Swarm Optimization (PSO)

Este algoritmo hace referencia al comportamiento de las bandadas de pájaros y cardúmenes de peces al realizar búsquedas óptimas [7]. Las partículas encuentran la solución a partir de su mejor posición y la mejor posición de todas las partículas [8]. Las ecuaciones que modelan esta situación son las siguientes [7], [8]:

$$v_{id}^{t+1} = v_{id}^{t+1} + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (p_{gd}^t - x_{id}^t) \quad (1)$$

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1} \quad (2)$$

Donde v es la velocidad, x la posición, c son constantes de aceleración, p_{id} es la mejor

posición personal, p_{gd} es la mejor posición que tuvo todo el enjambre en la iteración y r son números aleatorios entre 0 y 1. La fase de exploración ocurre cuando se tiene una diferencia grande entre la posición de la partícula y el particle best (pbest) o global best (gbest). La ventaja de este método contra los que necesitan usar el gradiente es que no se requiere que el problema sea diferenciable y puede optimizarse problemas con ruido o cambiantes. La función que se desea optimizar en este caso se llama función de costo [7].

6.2.1. Funcionamiento del algoritmo PSO

Se inicia el programa creando las partículas, dándoles una posición y velocidad inicial. Luego se introducen esos valores a la función de costo, se actualizan los valores de pbest y gbest. Este proceso se repite hasta que se cumpla un número de iteraciones o se logre la convergencia del algoritmo. La convergencia se alcanza cuando todas las partículas son atraídas a la partícula con la mejor solución (gbest). Un factor que se debe tomar en cuenta es el de restringir los valores que pueden tomar las funciones para que el algoritmo no diverja [8].

6.2.2. Mejora del algoritmo

El algoritmo puede mejorarse a través del incremento de partículas, introducción del peso de inercia w o la introducción de un factor de constricción K. El factor de inercia controla las fases de explotación y desarrollo del algoritmo. Se sugiere utilizar un valor mayor de inercia y decrementarlo hasta un valor menor para tener una mayor exploración al principio, pero al final mayor explotación [7].

$$v_{id}^{t+1} = wv_{id}^{t+1} + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (p_{gd}^t - x_{id}^t) \tag{3}$$

$$v_{id}^{t+1} = K(wv_{id}^{t+1} + c_1 r_1 (p_{id}^t - x_{id}^t) + c_2 r_2 (p_{gd}^t - x_{id}^t)) \tag{4}$$

El otro caso es utilizar el factor de constricción K para mejorar la probabilidad de convergencia y disminuir la probabilidad de que las partículas se salgan del espacio de búsqueda [7].

6.3. Ant Colony Optimization (ACO)

A partir de las observaciones de los entomólogos, se determinó que las hormigas tienen la habilidad de encontrar el camino más corto entre una fuente de alimento y su hormiguero. Marco Dorigo desarrolló un algoritmo con base en el comportamiento de estos insectos, a partir del cual se han derivado muchas otras variantes. Sin embargo, la idea principal de utilizar a las hormigas como base del algoritmo puede verse como una instancia de la metaheurística de Ant Colony Optimization (ACO-MH). Algunas de las instancias más conocidas son: Ant System (AS), Ant Colony System (ACS), Max-Min Ant Aystem (MMAS), Ant-Q, Fast Ant System, Antabu, AS-rank y ANTS [6].

Cuando una hormiga encuentra una fuente de alimento, al regresar al nido esta deja un rastro de feromonas para indicarle a sus compañeras que si siguen ese camino, encontrarán comida. Mientras más hormigas escojan ese camino, más fermona habrá y asimismo, mayor probabilidad de que las demás hormigas elijan esa ruta. Esta forma de comunicación indirecta que tienen las hormigas es denominada *stigmergy*. «La feromona artificial imita las características de la feromona real, indicando la popularidad de la solución del problema de optimización que se está resolviendo. De hecho, la feromona artificial funciona como una memoria a largo plazo del proceso completo de la búsqueda»[6].

Al principio, las hormigas se comportan de una manera aleatoria. Deneubourg realizó un experimento en el que se colocó comida a cierta distancia del hormiguero, pero solo podían acceder a ella en dos caminos diferentes. En este experimento, los caminos eran del mismo tamaño, pero con el comportamiento aleatorio de las hormigas uno de ellos fue seleccionado. Este experimento es conocido como el "puente binario". Asimismo, en un experimento similar pero con uno de los caminos más corto que el otro, las hormigas al principio eligieron los dos de forma equitativa, pero conforme el tiempo pasó eligieron el más corto. Este efecto es denominado "largo diferencial"[6].

6.3.1. Simple Ant Colony Optimization (SACO)

Este algoritmo es el que se utilizó en la implementación del experimento del puente binario. Este considera el problema general de búsqueda del camino más corto entre un conjunto de nodos en un grafo $G = (V, E)$ donde la matriz V representa a todos los vértices o nodos del grafo y la matriz E a todas las aristas o *«edges»* del grafo [6].

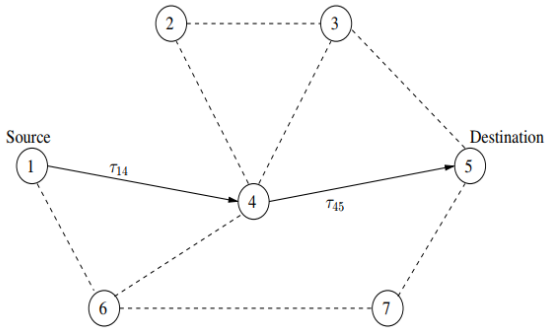

Figura 1: Ejemplo de un grafo [6].

Asimismo, el largo L^k del camino construido por la hormiga k se calcula como el número de saltos en el camino desde el nodo que representa al nido hasta el que representa el destino con la comida. En la figura 1 puede verse un camino marcado con flechas continuas con un largo de 2. En cada arista (i,j) del grafo se tiene cierta concentración de feromona asociada τ_{ij}. En este algoritmo, la concentración inicial de feromona se asigna de forma aleatoria[1] (aunque en la vida real sea de 0). Cada hormiga decide inicialmente de forma aleatoria[2] a qué arista dirigirse. K hormigas se colocan en el nodo fuente (source). Para cada iteración, cada

[1]Sin embargo en [9] recomiendan que todas las aristas comiencen con el mismo valor, como 1 por ejemplo.
[2]En [9] y [6] recomiendan usar el algoritmo Roulette Wheel.

hormiga construye una solución al nodo destino. En cada nodo i, cada hormiga k determina a qué nodo j debe de dirigirse basado en la probabilidad p:

$$p^k_{(i,j)}(t) = \begin{cases} \frac{(\tau_{ij}(t))^\alpha}{\sum_{k \in J_k} (\tau_{ij}(t))^\alpha} & \text{si j} \in N^k_i \\ 0 & \text{si j} \notin N^k_i \end{cases} \tag{5}$$

Donde N representa al conjunto de nodos viables conectados al nodo i. Si en cualquier nodo el conjunto N es el conjunto vacío, entonces el nodo anterior se incluye. En la ecuación 5, α es una constante positiva que se utiliza para modular la influencia de la concentración de feromona. Cuando este parámetro es muy grande la convergencia puede ser extremadamente rápida y resultar en un camino no óptimo. Cuando las hormigas llegan al nodo destino regresan a su nodo fuente por el mismo camino por el cual llegaron y depositan feromona en cada arista [6]. En este proceso también se modela una forma de evaporación de feromona para aumentar la probabilidad de exploración del terreno.

La ecuación que gobierna la evaporación de la feromona en los trayectos es la siguiente:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} (\Delta\tau^k_{ij}(t)) \tag{6}$$

Donde m es el número de hormigas, ρ es el factor de evaporación de feromona (entre 0 y 1). Además, se sabe que:

$$\Delta\tau^k_{ij}(t) = \frac{Q}{L_k}(t) \tag{7}$$

Donde Q es una constante y L es el costo del trayecto, como el largo del mismo. El valor representa el cambio de feromona entre el nodo i y j que la hormiga visitó en la iteración t [7].

6.3.2. Ant System

Este fue el primer algoritmo desarrollado que emula el comportamiento de las hormigas, que consiste en hacer ciertas mejoras al algoritmo presentado anteriormente. El algoritmo anterior tenía un objetivo más instructivo, por lo que era más simple. Algunas de las mejoras son la incorporación de información heurística en la ecuación de probabilidad y agregando capacidad de memoria con una lista tabú. La nueva ecuación de probabilidad es la siguiente:

$$p^k_{(i,j)}(t) = \begin{cases} p^k_{(i,j)}(t) = \frac{(\tau_{ij}(t))^\alpha (\eta_{ij}(t))^\beta}{\sum_{k \in J_k} (\tau_{ij}(t))^\alpha (\eta_{ij}(t))^\beta} & \text{si j} \in N^k_i \\ 0 & \text{si j} \notin N^k_i \end{cases} \tag{8}$$

Donde el nuevo parámetro η representa el inverso del costo de la arista. En ambos algoritmos presentados, la condición de paro puede ser un límite de iteraciones o generaciones, o bien, cuando un porcentaje determinado de hormigas haya elegido la misma solución.

6.4. Grafos

Como pudo observarse anteriormente, el algoritmo Ant System es un algoritmo de búsqueda basado en grafos y no funciones como el PSO. Por lo tanto, es necesario definir qué es un grafo y de qué maneras puede representarse. Un grafo G es un par de conjuntos (V, E) donde V representa al conjunto de vértices del grafo y E al conjunto de pares no ordenados de elementos de V [10].

$$V = \{V_1, V_2, \dots V_n\}$$
$$E = \{(V_i, V_j), (V_i, V_j), \dots\}$$

Existen los grafos dirigidos (o digrafos), donde el enlace del nodo i al j no es el mismo que del nodo j al i. Estos grafos sí cuentan con dirección. Asimismo, existen los grafos (y digrafos) ponderados. A estos se les asocia un número de peso o costo a cada arista. Este número podría ser distancia, capacidad o valor temporal dependiendo de la aplicación [11].

Los grafos pueden representarse a partir de dibujos como el de la figura 1, donde los grafos se muestran como en la figura y los digrafos con flechas para indicar la dirección. Además, también pueden representarse a partir de matrices como la matriz de adyacencia, En esta matriz cuadrada de tamaño número de vértices se muestra qué vértices están y no están conectados con un 1 y un 0 respectivamente [12]. A continuación se muestra un ejemplo de matriz de adyacencia de la figura 1.

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Como puede observarse, cada fila y columna representa a un vértice, empezando desde el vértice 1 en la primera fila y primera columna (esquina superior izquierda).

6.4.1. Grafos en Matlab

Para representar grafos en Matlab se utiliza la clase *graph*, ya disponible para utilizarse en Matlab. Estos objetos de la clase graph tienen métodos para acceder y modificar nodos

y aristas, para mostrar el grafo en representación de matrices, mostrar información del grafo, visualización e incluso para encontrar el camino más corto entre nodos. Algunos de los atributos (o propiedades como les llama Matlab) son las aristas y los nodos, aunque es posible agregarle más atributos a la clase. Para mas información y documentación acerca de la clase *graph* de Matlab visitar [13].

6.5. Programación orientada a objetos

Como se explicó anteriormente, el software Matlab representa los grafos como objetos, por lo que es necesario saber lo básico de cómo funcionan estos para implementarlos. Primero, ¿Qué es un objeto? Un objeto es una colección de datos con ciertos comportamientos asociados. Un objeto puede ser de una clase en específico, que se define como una plantilla para crear objetos. El objeto en sí de una clase se llama instancia, y es el que se utiliza activamente en la programación. Para diferenciar objetos se utilizan atributos o características del mismo. Por ejemplo, un objeto de la clase fruta puede ser manzana o pera y se diferencian por sus atributos color y forma (por ejemplo) [14].

Como se explicó en el párrafo anterior, un objeto tiene comportamientos asociados, lo que nos hace pensar en ciertas funciones o acciones que pueden hacerse con el objeto. Estas funciones se llaman métodos y básicamente son funciones que solo pueden ser utilizadas por instancias de la clase donde están definidas [14]. Generalmente para referirse a un atributo se utiliza la notación «instancia.atributo» y para usar un método se utiliza «instancia.función()». Para conocer y aprender sobre la notación específica de programación orientada a objetos en Matlab puede verse [15].

6.6. Programación paralela

Para realizar pruebas debe de correrse el programa varias veces, por lo que se consumiría demasiado tiempo dependiendo de la cantidad y forma de realizar las pruebas. Por lo tanto se propone utilizar programación paralela para aprovechar los distintos cores de las computadoras para acelerar el proceso [16].

Los softwares tradicionales se escriben para computación serial y no paralela, por lo que el problema se divide en pedazos pequeños y se van pasando uno por uno al procesador como puede verse en la figura 2 [16].

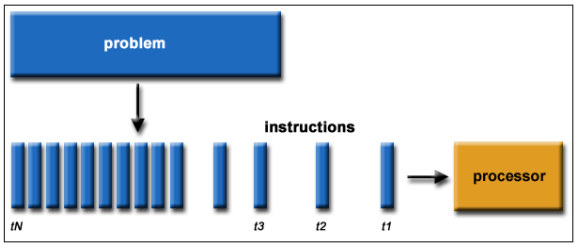

Figura 2: Computación serial [16]

Sin embargo, en la computación paralela la tarea se divide en el número de cores que tenga la máquina para realizar tareas en simultáneo y así ahorrar tiempo de ejecución (ver figura 3). Además, con paralelización es posible resolver problemas más complejos y al minimizar el tiempo (dependiendo de la aplicación) incluso podría ahorrarse algún costo [16].

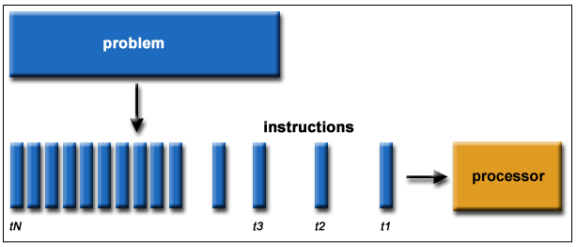

Figura 3: Computación paralela [16]

6.6.1. Programación paralela en Matlab

Matlab tiene un Toolbox de computación paralela con funciones como el *parfor*, que reemplaza el ciclo for normal por un ciclo for paralelizado. Para correr un programa paralelizado de Matlab primero es necesario habilitar el «Parallel Pool» de Matlab y establecer la cantidad de procesadores (Matlab les llama workers) a utilizar [16].

Para re-escribir un for normal a un *parfor* se requiere que cada una de las sub-tareas sea independiente, por lo que no pueden utilizarse entre sí. Además, también es necesario que el orden de ejecución de las tareas no importe (también el orden debe ser independiente). Algunas restricciones para el parfor son que no se puede introducir variables utilizando las funciones load, eval y global, no pueden contener break o return y tampoco puede contener otro parfor dentro [16]. Para más información sobre este tema se recomienda ver [17].

Además de la programación paralela en Matlab, se recomienda utilizarla en conjunto con buenas prácticas de programación [18] y técnicas específicas para mejorar el desempeño [19], que incluyen estrategias como vectorización de código [20]. Si llegara a pasar que el código está trabajando de forma muy lenta es posible utilizar la técnica de búsqueda de cuellos de botella dada en [21] específicamente para Matlab utilizando la función *profiler*. Si se desea implementar paralelización en Matlab se recomienda leer todas las referencias bibliográficas de esta sección.

6.7. Planificación de movimiento

Es el problema de encontrar el movimiento desde un destino hasta un final esquivando obstáculos (si hubiese) y cumpliendo restricciones de juntas y/o torque para un robot [22].

6.7.1. Espacio de configuración

6.8. Grafos de visibilidad

6.9. Probabilistic Road Maps

6.10. Random R Trees

6.11. Robots diferenciales

6.12. Controladores de posición y velocidad de robots diferenciales

CAPÍTULO 7

Resultados

7.1. Simulación simple de AS

Primero lo que hice fue codificar el Simple Ant Colony y luego sobreescribirlo con el Ant System, que es un modelo un poco más complejo pues toma en consideración el costo por distancia y no solo la cantidad de feromona depositada. Además, este algoritmo también toma en cuenta otra variable para escalar la cantidad de feromona que se deposita. A continuación se presenta el pseudogódigo que se siguió, un camino óptimo y una imagen de su simulación de feromona, así como una imagen de la falla del algortimo SACO sin tener una lista tabu (otra mejora que tiene AS0). Luego se presenta un resumen de las variables utilizadas y algunos resultados de corridas realizadas.

Algorithm 17.3 Ant System Algorithm

$t = 0$;
Initialize all parameters, i.e. $\alpha, \beta, \rho, Q, n_k, \tau_0$;
Place all ants, $k = 1, \ldots, n_k$;
for *each link* (i, j) **do**
    $\tau_{ij}(t) \sim U(0, \tau_0)$;
end
repeat
    for *each ant* $k = 1, \ldots, n_k$ **do**
        $x^k(t) = \emptyset$;
        repeat
            From current node i, select next node j with probability as defined in equation (17.6);
            $x^k(t) = x^k(t) \cup \{(i, j)\}$;
        until *full path has been constructed*;
        Compute $f(x^k(t))$;
    end
    for *each link* (i, j) **do**
        Apply evaporation using equation (17.5);
        Calculate $\Delta\tau_{ij}(t)$ using equation (17.10);
        Update pheromone using equation (17.4);
    end
    for *each link* (i, j) **do**
        $\tau_{ij}(t + 1) = \tau_{ij}(t)$;
    end
    $t = t + 1$;
until *stopping condition is true*;
Return $x^k(t) : f(x^k(t)) = \min_{k'=1,\ldots,n_k}\{f(x^{k'}(t))\}$;

Figura 4: Pseudocódigo del algoritmo [6].

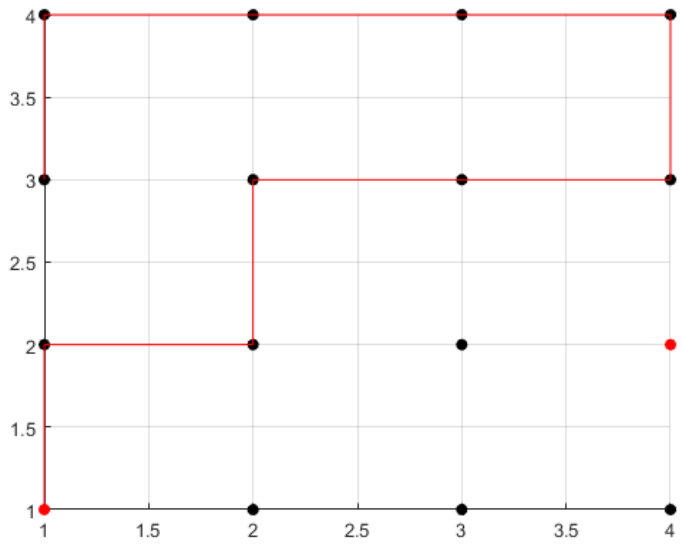

Figura 5: Algoritmo sin retorno (sin lista tabu).

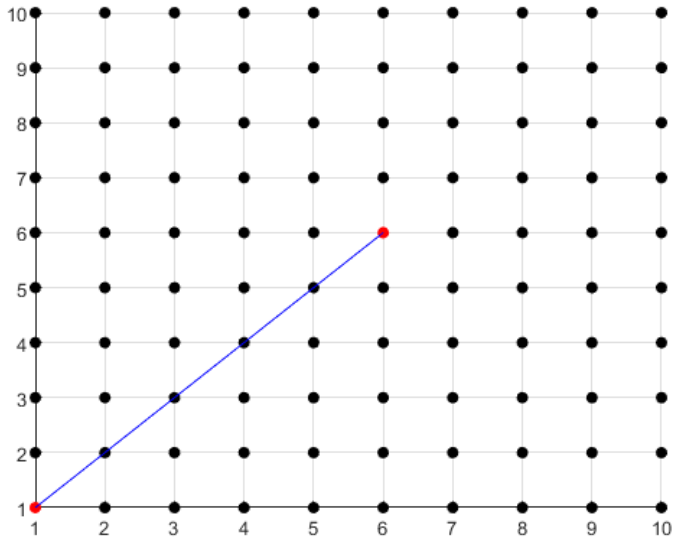

Figura 6: Camino óptimo encontrado del nodo (1,1) al (6,6).

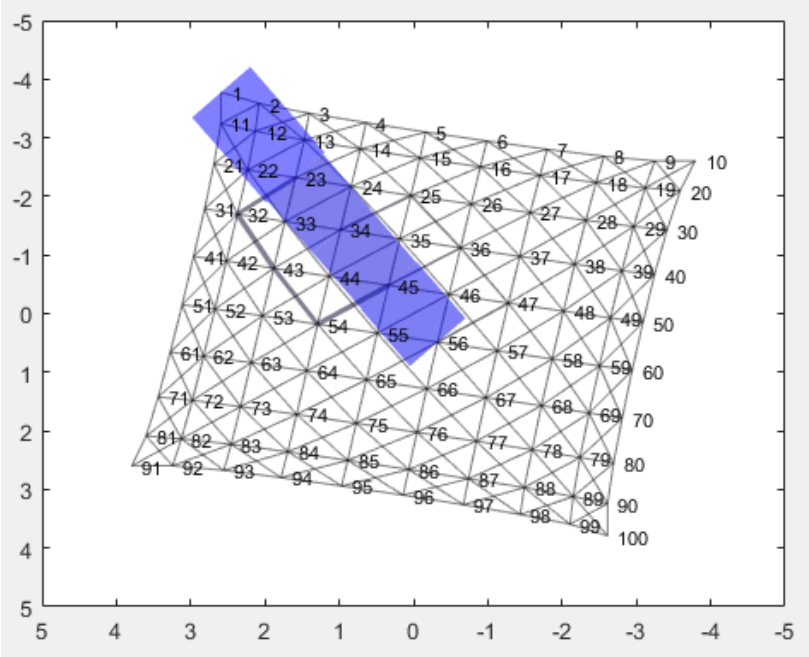

Figura 7: Feromona del camino óptimo encontrado del nodo (1,1) al (6,6).

7.2. Variables del código

Las variables en negrilla representan a variables tipo celda, de lo contrario la variable es un array o matriz normal. Asumiendo que tenemos k hormigas[1], n nodos en el grafo y un número máximo de iteraciones tf:

Matrices de adyacencia con pesos: mientras más grandes sean los valores, mayor probabilidad tendrán de ser escogidos por la función ant_decision.

$$\tau = \begin{bmatrix} \tau_{11} & \dots & \tau_{1n} \\ \vdots & \ddots & \vdots \\ \tau_{n1} & \dots & \tau_{nn} \end{bmatrix} \qquad \eta = \begin{bmatrix} \eta_{11} & \dots & \eta_{1n} \\ \vdots & \ddots & \vdots \\ \eta_{n1} & \dots & \eta_{nn} \end{bmatrix}$$

La siguiente variable es una celda que contiene vectores columna:

$$\boldsymbol{vytau} = \begin{bmatrix} \text{vecinos nodo 1} & \tau \text{ de los vecinos nodo 1} \\ \vdots & \vdots \\ \text{vecinos nodo n} & \tau \text{ de los vecinos nodo n} \end{bmatrix}$$

A partir de ahora utilizaremos la variable x para referirnos a vectores fila (x,y) apilados en una columna. En feas nodes se guardan los nodos no visitados o «viables» a los que cada

[1] en el código el número se expresa como «hormigas», pero por términos de simplificación se utilizará k en este documento

nodo sí puede dirigirse. En la celda las node se guarda los nodos visitados anteriormente.

$$\boldsymbol{feas\_nodes} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{k1} & \dots & x_{kn} \end{bmatrix} \qquad \boldsymbol{last\_node} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{k1} & \dots & x_{kn} \end{bmatrix}$$

Blocked nodes guarda la lista de nodos ya visitados por cada nodo y path k guarda la lista de los nodos escogidos por la hormiga k (la actual).

$$\boldsymbol{blocked\_nodes} = \begin{bmatrix} x_{11} \\ \vdots \\ x_{k1} \end{bmatrix} \qquad \boldsymbol{path\_k} = \begin{bmatrix} x_{11} \\ \vdots \\ x_{k1} \end{bmatrix}$$

L guarda el costo total de cada path por hormiga (filas) y por iteración (columnas), mientras que all path guarda cada path (vectores fila apilados) por hormiga (filas) y por iteración (columnas).

$$L = \begin{bmatrix} L_{11} & \dots & L_{1t} \\ \vdots & \ddots & \vdots \\ L_{k1} & \dots & L_{kt} \end{bmatrix} \qquad \boldsymbol{last\_node} = \begin{bmatrix} x_{11} & \dots & x_{1t} \\ \vdots & \ddots & \vdots \\ x_{k1} & \dots & x_{kt} \end{bmatrix}$$

7.3. Parámetros utilizados

Se utilizó el nodo (6,6) como nodo de prueba. Los parámetros son sensibles a la lejanía de los nodos, por lo que se recomienda hacer el barrido de parámetros con respecto a la lejanía con la que se desea probar el algoritmo. Para esta entrega se utilizó los siguientes parámetros:

Parámetro	**Valor**
ρ	0.5
α	1.3
β	1
Q	1
tf	70
ϵ	0.9
τ_0	0.1

Realicé 10 corridas a mano, dado que mi computadora está pasando por momentos difíciles y no puede correr muchos experimentos. Los resultados se muestran en la tabla a continuación:

En este ejemplo el costo mínimo que se podía obtener era 2.5, ya que el costo de las diagonales era de 0.5. El tiempo fue medido con las funciones tic y toc de Matlab. Al final

t	tiempo	costo
14	14.2161	2.5
10	9.6620	2.5
9	9.2031	2.5
18	49.5735	3.5
43	44.7586	3.5
17	16.0491	2.5
26	25.3741	3.5
11	26.8102	2.5
22	60.7632	3.5
13	12.3290	3.5
18.3	**26.8739**	**3**

de la tabla se muestra la media de cada parámetro. Para cada corrida es posible observar que el tiempo aumenta cuando las hormigas toman malas decisiones al principio, puede que esto se arregle modificando la tasa de evaporación para aumentar el tiempo de exploración de las hormigas.

CAPÍTULO 8

Conclusiones

CAPÍTULO 9

Recomendaciones

Recomiendo abrir una línea de investigación sobre Ant algorithms para aplicaciones de VLSI

CAPÍTULO 10

Bibliografía

[1] A. S. A. Nadalini, «Algoritmo Modificado de Optimización de Enjambre de Partículas (MPSO)», Universidad del Valle de Guatemala, UVG, nov. de 2019.

[2] *Robotarium | Institute for Robotics and Intelligent Machines*. dirección: `http://www.robotics.gatech.edu/robotarium`.

[3] *Programmable Robot Swarms*, en-US, ago. de 2016. dirección: `https://wyss.harvard.edu/technology/programmable-robot-swarms/` (visitado 06-04-2020).

[4] S. P. ROUL, «Application of Ant Colony Optimization for finding Navigational Path of mobile robot», National Institute of Technology, Rourkela, 2011.

[5] *Comparative Analysis of Ant Colony and Particle Swarm Optimization Techniques V.Selvi Lecturer, Department of Computer Science, Nehru Memorial College,*

[6] J. Wang y A. Kusiak, eds., *Computational intelligence in manufacturing handbook*, en, ép. Mechanical engineering. Boca Raton, FL: CRC Press, 2001, ISBN: 978-0-8493-0592-4.

[7] M. N. A. Wahab, S. Nefti-Meziani y A. Atyabi, «A Comprehensive Review of Swarm Optimization Algorithms», *PLoS ONE*, vol. 10, n.º 5, 2015. DOI: `https://doi.org/10.1371/journal.pone.0122827`.

[8] Y. Zhang, S. Wang y G. Ji, «A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications», *Hindawi*, vol. 2015, n.º 931256, 2015. DOI: `http://dx.doi.org/10.1155/2015/931256`.

[9] M. Dorigo y T. Stützle, *Ant colony optimization*, en. Cambridge, Mass: MIT Press, 2004, ISBN: 978-0-262-04219-2.

[10] C. A. Maeso, *Métodos basados en grafos*. dirección: `http://pdg.cnb.uam.es/pazos/cursos/bionet_UAM/Grafos_CAguirre.pdf`.

[11] K. Thulasiraman y M. N. S. Swamy, *Graphs: Theory and Algorithms*, en. John Wiley & Sons, mar. de 2011, Google-Books-ID: rFH7eQffQNkC, ISBN: 978-1-118-03025-7.

[12] K. " Thulasiraman, S. Arumugam, A. Brandstädt y T. Nishizeki, eds., *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*, en, 0.ª ed. Chapman y Hall/CRC, ene. de 2016, ISBN: 978-0-429-15023-4. DOI: 10.1201/b19163. dirección: https://www.taylorfrancis.com/books/9781420011074 (visitado 12-07-2020).

[13] *Graph with undirected edges - MATLAB - MathWorks América Latina.* dirección: https://la.mathworks.com/help/matlab/ref/graph.html (visitado 12-07-2020).

[14] D. Phillips, *Python 3 object oriented programming: harness the power of Python 3 objects*, en, ép. Community experience distilled. Birmingham: Packt Publ, 2010, OCLC: 802342008, ISBN: 978-1-84951-126-1.

[15] *Clases - MATLAB & Simulink - MathWorks América Latina.* dirección: https://la.mathworks.com/help/matlab/object-oriented-programming.html (visitado 12-07-2020).

[16] T. Mathieu, G. Hernandez y A. Gupta, «Parallel Computing with MATLAB», en, pág. 56,

[17] *Using parfor Loops: Getting Up and Running*, Library Catalog: blogs.mathworks.com. dirección: https://blogs.mathworks.com/loren/2009/10/02/using-parfor-loops-getting-up-and-running/ (visitado 13-07-2020).

[18] *Programming Patterns: Maximizing Code Performance by Optimizing Memory Access*, en, Library Catalog: la.mathworks.com. dirección: https://la.mathworks.com/company/newsletters/articles/programming-patterns-maximizing-code-performance-by-optimizing-memory-access.html (visitado 13-07-2020).

[19] *Techniques to Improve Performance - MATLAB & Simulink - MathWorks América Latina.* dirección: https://la.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html (visitado 13-07-2020).

[20] *Vectorization - MATLAB & Simulink - MathWorks América Latina.* dirección: https://la.mathworks.com/help/matlab/matlab_prog/vectorization.html (visitado 13-07-2020).

[21] *Buscar cuellos de botella de código - MATLAB & Simulink - MathWorks América Latina.* dirección: https://la.mathworks.com/help/matlab/creating_plots/assessing-performance.html (visitado 13-07-2020).

[22] K. M. Lynch y F. C. Park, *Modern robotics: mechanics, planning, and control*, en. Cambridge, UK: Cambridge University Press, 2017, OCLC: ocn983881868, ISBN: 978-1-107-15630-2 978-1-316-60984-2.

CAPÍTULO 11

Anexos

11.1. Repositorio de Github

CAPÍTULO 12

Glosario

fórmula Una expresión matemática. 9

latex Es un lenguaje de marcado adecuado especialmente para la creación de documentos científicos. 9