EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO EINVESTIGACIÓN

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"POR MI PATRIA Y POR MI BIEN"

TESIS

ANÁLISIS COMPARATIVO DEL DESEMPEÑO DE METAHEURÍSTICAS IMPLEMENTADASEN HARDWARE

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

Presenta:
Ing. Yazef Valerio Espinosa
G13071433

Director de Tesis:
Dra. Claudia Guadalupe Gómez Santillán

Co-Director de Tesis:
Dr. Nelson Rangel Valdez

Cd. Madero, Tamps. Méx. junio 2021

Cd. Madero, Tam. 18 de mayo de 2021

OFICIO No. : U.030/21
ASUNTO: AUTORIZACIÓN DE IMPRESIÓN DE TESIS

C. YAZEF VALERIO ESPINOSA
No. DE CONTROL G13071433
P R E S E N T E

Me es grato comunicarle que después de la revisión realizada por el Jurado designado para su Examen de Grado de Maestría en Ciencias de la Computación, se acordó autorizar la impresión de su tesis titulada:

"ANÁLISIS COMPARATIVO DEL DESEMPEÑO DE METAHEURÍSTICAS IMPLEMENTADAS EN HARDWARE"

El Jurado está integrado por los siguientes catedráticos:

PRESIDENTE:	DR. HÉCTOR JOAQUÍN FRAIRE HUACUJA
SECRETARIO:	DRA. LAURA CRUZ REYES
VOCAL:	DRA. CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
SUPLENTE:	DR. NELSON RANGEL VALDEZ
DIRECTOR DE TESIS:	DRA. CLAUDIA GUADALUPE GÓMEZ SANTILLÁN
CO-DIRECTOR DE TESIS:	DR. NELSON RANGEL VALDEZ

Es muy satisfactorio para la División de Estudios de Posgrado e Investigación compartir con usted el logro de esta meta. Espero que continúe con éxito su desarrollo profesional y dedique su experiencia e inteligencia en beneficio de México.

A T E N T A M E N T E
Excelencia en Educación Tecnológica.
"Por mi patria y por mi bien".

MARCO ANTONIO CORONEL GARCÍA
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
INSTITUTO TECNOLÓGICO DE CIUDAD MADERO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

c.c.p.- Archivo
MACG 'mdcoa'

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) y a la Dirección General de Educación Tecnológica (DGEST) por el apoyo y facilidades recibidas para la conclusión de este trabajo.

De igual manera agradezco al Instituto Tecnológico de Cd. Madero por las facilidades proporcionadas para el desarrollo de este proyecto, así como a los profesores que contribuyeron a mi formación académica.

Reciban un profundo agradecimiento los integrantes del comité tutorial de esta tesis: Dra. Laura Cruz Reyes, Dr. Nelson Rangel Valdéz, Dra. Claudia Guadalupe Gómez Santillán, Dra. María Lucila Morales Rodríguez. Siempre tendré en mente sus consejos.

Le agradezco infinitamente a mi asesora de tesis, la Dra. Claudia Guadalupe Gómez Santillán, por todo el apoyo que me brindó durante la maestría. Mi respeto y admiración por su dedicación y enseñanza. Es una mujer inteligente, trabajadora y cálida.

Mi agradecimiento más especial es para las dos personas más importantes de mi vida, mis padres por su entusiasmo por la familia, amor incondicional, confianza en mí y toda su enseñanza hacia mí. En cada etapa de mi vida, sus consejos me han hecho mejor, y me enseñó a luchar por mis sueños. Siempre estaré agradecido por todo lo que me den.

Resumen

Las tarjetas de desarrollo tienen como principal objetivo servir como una herramienta para el desarrollo de sistemas integrados, tales como relojes inteligentes, sistemas de proposito general, automatización de procesos, robots de diferentes propositos, domotica, etc, que permiten solucionar problemas del mundo real, en donde los procesos no demanden el uso de recursos elevados, debido a las limitaciones presentadas en las tarjetas de desarrollo.

Las tarjetas de desarrollo pueden trabajar en conjunto con sensores y actuadores, en donde los primeros son capaces de recoger información del entorno y los segundos realizar una acción de respuesta inmediata de acuerdo con las necesidades del usuario, además que se pueden conectar con otras tarjetas de desarrollo para reducir la cantidad de procesos manteniendo un buen rendimiento en los tiempos de respuesta y/o resultados.

En el presente trabajo de tesis se formula una adaptación basado en el problema clásico del Traveling Salesman Problem (TSP) en donde se busca obtener las rutas de ciudades con el menor costo, cumpliendo con las restricciones del problema, dicha adaptación es resuelta bajo la implementación de un vehiculo móvil desarrollado por dos tarjetas de desarrollo, Arduino y Raspberry Pi, al cual se le agregaron sensores y actuadores que permitieran su desplazamiento por el entorno, ha dicho dispositivo se le implementó dos metaherusiticas, un algoritmo genético y un algoritmo colonia de hormigas, las cuales dan solución al problema propuesto.

Se propone una metodología para analizar las características de las tarjetas de desarrollo con el fin de observar las limitaciones y características que las afectan. El método incluye tres etapas: la primera etapa determina la entrada de datos que se utilizará para resolver el problema. La segunda fase incluye la arquitectura desarrollada para cada tarjeta. El último es obtener los resultados, para analizar y escribir el comportamiento de la tarjeta de desarrollo.

Summary

The main objective of the development cards is to serve as a tool for the development of integrated systems, such as smart watches, general-purpose systems, process automation, robots for different purposes, domotics, etc., which allow solving computer problems. real world, where the processes do not demand the use of high resources, due to the limitations presented in the development cards.

The development cards can work in conjunction with sensors and actuators, where the former are capable of collecting information from the environment and the latter perform an immediate response action according to the user's needs, in addition to being able to connect with other cards. development to reduce the number of processes while maintaining good performance in response times and / or results.

In this thesis work, an adaptation based on the classic problem of the Traveling Salesman Problem (TSP) is formulated, in which it is sought to obtain the routes of cities with the lowest cost, complying with the restrictions of the problem, said adaptation is resolved under the implementation of a vehicle Mobile phone developed by two development boards, Arduino and Raspberry Pi, to which sensors and actuators were added to allow it to move around the environment, said device has implemented two metaherusitics, a genetic algorithm and an ant colony algorithm, which provide a solution to the problem. proposed.

A methodology is proposed to analyze the characteristics of development cards in order to observe the limitations and characteristics that affect them. The method includes three stages: the first stage determines the data input that will be used to solve the problem. The second phase includes the architecture developed for each card. The last is to get the results, to analyze and write the behavior of the development board.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Con el avance tecnológico se ha buscado brindar un apoyo en las actividades diarias de los seres humanos,tales como llamar por telefono, realizar videollamadas con distintas personas al rededor del mundo, activar aplicaciones para despertarse temprano, etc, dando como resultado el consumo de grandes cantidades de recursos de software y hardware. Debido a esto se ha incrementado la necesidad de crear nuevos dispositivos que sean sencillos de entender y de adquirir, que puedan ser capaces de adaptarse fácilmente a las necesidades de las personas.

Las tarjetas de desarrollo son herramientas destacadas para la programación e innovación de dispositivos dedicados a una tarea específica en conjunto con otros elementos, como son sensores, actuadores e incluso otras tarjetas de desarrollo. Debido a esto existe una gran variedad de tipos y modelos empleadas para diferentes problemas, las cuales poseen diversas características como la portabilidad, el bajo costo dentro del mercado y el uso multiplataforma.

Sin embargo, presentan características asociadas con la limitación de recursos, como es la capacidad de memoria y la velocidad con la que procesan los datos, por lo que es necesario hacer un estudio que permita identificar el comportamiento de diferentes modelos de tarjetas de desarrollo cuando son empleadas bajo un mismo problema a resolver.

En este trabajo se definirán como tarjetas de desarrollo a aquellas arquitecturas que trabajan con un microcontrolador o microprocesador, como son arduino y raspberry pi, las cuales son usadas para la construcción de dispositivos inteligentes con los que se pueden interactuar y controlar procesos del mundo real.

A través de este proyecto de investigación se busca caracterizar el desempeño de algoritmos metaheurísticos que solucionen un problema de optimización implementados en dos tarjetas de desarrollo, arduino y rasperry pi, para llevar a cabo un análisis comparativo que nos permita describir y evaluar el desempeño de las tarjetas de desarrollo a través de los resultados obtenidos.

1.1. Problema de investigación

En este trabajo se tomará como base el problema del problema del agente viajero (TSP) el cual será adaptado a uno más específico para poder trabajar con las tarjetas de desarrollo llamado TSP-RA, en el cual existe un conjunto de n ciudades, $V = \{1, 2, 3, \ldots, n\}$ y un conjunto de caminos uniendo cada una de las ciudades, así el camino $(i, j) \in A$, c_{ij} es la distancia para ir de la ciudad i a la ciudad j. Un agente viajero debe realizar un tour comenzando en una ciudad de origen y luego visitar todas las otras ciudades una única vez y retomar a la ciudad de origen. El problema consiste en hallar el recorrido de distancia mínima evitando subrecorridos. La formulación matemática del problema queda establecida como a continuación se describe.

$$minz(x) = \sum_{(i,j)\in A} c_{i,j}x_{i,j} \tag{1.1}$$

s.a.

$$\sum_{i:(i,j)\in A} x_{i,j} = 1 \qquad \forall j \in V \tag{1.2}$$

$$\sum_{j:(i,j)\in A} x_{i,j} = 1 \qquad \forall i \in V \tag{1.3}$$

$$\sum_{(i,j)\in A: i\in U, j\in (V-U)} x_{i,j} \geq 1 \qquad \forall S \subset V \tag{1.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A$$

Esta formulación fue propuesta por Dantzig, Fulkerson y Johnson (Dantzig, 1954). Las variables binarias x_{ij} indican si el arco (i, j) es utilizado en la solución. La función objetivo (1.1) establece que el costo total de la solución es la suma de los costos de los arcos utilizados. El modelo se encuentra sujeto a que en cada ruta se debe visitar un nodo una sola vez, estas restricciones se representan matemáticamente a través de las ecuaciones (1.2), (1.3), (1.4). Por su parte, las ecuaciones (1.2) y (1.3) garantizan que los puntos i y j, se visiten una sola vez en cada ruta. Finalmente, la restricción (1.4) es llamada restricciones de eliminación de subrecorridos e indica que todo subconjunto de nodos S debe ser abordado al menos una vez.

El modelo antes mencionado es adaptado con los elementos descritos en las Tablas 1.1 y 1.2, donde en la primera se describen las variables utilizadas en el proceso de solución, en la segunda de muestra los elementos que se utilizan un dispositivo móvil como el agente viajero. El dispositivo móvil estará en una habitación con objetos separados entre sí, lo cuales se toman como nodos. La distancia entre cada nodo es el camino que puede tomar el dispositivo móvil.

Tabla 1.1: Variables de definición del problema propuesto TSP-RA.

Variable	Definición
inicio_VM	Inicio del tiempo del vehículo al empezar el ruteo.
fin_VM	Fin del tiempo del vehículo al terminar el ruteo.
sensor_1	Variable binaria donde se registra si se utiliza el sensor 1.
sensor_2	Variable binaria donde se registra si se utiliza el sensor 2.
MAX_tiempo	Variable con el tiempo máximo de ruteo.
x_ruta	Estructura donde se va registrando que nodo se ha visitado.
x_r	Variable donde se compara si un elemento en *x_ruta* ha sido visitado.
x_tiempo	Estructura donde se va registrando el tiempo del nodo que se ha visitado
x_t	Variable donde se compara si un elemento en *x_tiempo* ha sido visitado

El dispositivo debe encontrar la ruta mínima que visite una única vez a todos los nodos, para esto se propone implementar algoritmos para la solución del problema del TSP, los cuales estarán desarrollados en las dos tarjetas de desarrollo. Por lo cual, el planteamiento del problema denominado TSP-RA se establece de la siguiente manera.

Tabla 1.2: Tabla de correspondencia del problema propuesto TSP-RA

TSP	Adaptación al problema establecido	Variables
Agente viajero	Dispositivo móvil	t
Ciudades	Nodos	n
Caminos	Caminos	d
Función objetivo	Función objetivo	$z(d)$
	Sensor 1	s1
	Sensor 2	s2

Existe un conjunto de n nodos, $N = \{1, 2, 3, \ldots, n\}$ y un conjunto de caminos, $D = \{1, 2, 3, \ldots, d\}$ uniendo cada una de los nodos, así el camino $d_{(i,j)} \in D$ es la distancia para ir del nodo i al nodo j. El dispositivo móvil debe realizar un tour comenzando en un nodo de origen y luego visitar todos lo de más nodos una sola vez. El problema consiste en hallar el recorrido de distancia mínima evitando subrecorridos, la expresión matemática del problema se establece de la siguiente manera.

$$minz(t) = \sum_{(i,j)\in N} c_{i,j} d_{i,j} \tag{1.5}$$

s.a.

$$\sum_{i:(i,j)\in D} d_{i,j} = 1 \qquad \forall j \in N \tag{1.6}$$

$$\sum_{j:(i,j)\in D} d_{i,j} = 1 \qquad \forall i \in N \tag{1.7}$$

$$\sum_{(i,j)\in D} d_{i,j} \geq 1 \qquad \forall S \subset N \tag{1.8}$$

$$\sum_{(i,j)\in D} s1_{i,j} \geq 1 \tag{1.9}$$

$$\sum_{(i,j)\in D} s2_{i,j} \geq 1 \tag{1.10}$$

$$d_{ij} \in \{0, 1\} \qquad \forall (i, j) \in N \tag{1.11}$$

Donde las ecuaciones (1.6), (1.7) y (1.8) siguen teniendo la misma función que en el problema de TSP, adicionalmente las ecuaciones (1.9) y (1.10) corresponden a que los sensores deben ser utilizados más de una vez.

El objetivo al resolver este problema es encontrar un conjunto de factores críticos que permitan caracterizar el desempeño de algoritmos que solucionen este problema, implementados en dos tarjetas de desarrollo para llevar acabo un análisis comparativo que nos permita describir y evaluar el desempeño de las tarjetas a través de los resultados obtenidos.

1.2. Justificación

Con los avances tecnológicos se han desarrollado sistemas inteligentes capaces de facilitar la obtención y manipulación de datos, con el fin de ser utilizada para beneficio del ser humano, dichos sistemas inteligentes están compuestos por tarjetas de desarrollo que hacen uso de sensores y actuadores, ya que el uso de estos es fundamental para interactuar con el entorno y poder medir variables, ingresar datos o realizar acciones físicas.

Debido a esto es necesario saber cuál es desempeño de las tarjetas cuando le son implementadas sensores y actuadores a la hora de su funcionamiento en tiempo real, ya que sus características técnicas son limitadas y además se cuenta con una gran variedad de modelos con diferentes funciones y tareas especificas.

Es por ello que en este trabajo de investigación se propone el análisis del desempeño de arquitecturas en funcionamiento para la solución de problemas, donde dichas tarjetas permiten la incorporación de sensores y actuadores, para detallar un conjunto de características apropiadas al hacer uso de estas.

1.3. Objetivos

A continuación, se presentan el objetivo general y los objetivos específicos planteados para esta investigación.

Objetivo General

Caracterizar el desempeño de algoritmos en la solución del problemas de optimización a través de tarjetas de desarrollo.

Objetivos específicos

- Caracterizar las arquitecturas de hardware arduino y raspberry pi.
- Caracterizar un caso de estudio donde se implementen ambas arquitecturas de hardware para la solución del problema de optimización
- Desarrollar una arquitectura que integre algoritmos, sensores y actuadores a las tarjetas de desarrollo para la solución del problemas de optimización.
- Medir la calidad del desempeño de los algoritmos.

1.4. Alcances y limitaciones

Es previsible que para la implementación del proyecto se deban considerar los siguientes alcances y limitaciones:

a) El problema a resolver es un problema de optimización.

b) Se utilizarán únicamente dos tarjetas de desarrollo; arduino UNO y raspberry pi 3B+.

c) Se implementará al menos un algoritmo metaheurístico.

d) El agente móvil será probado en una habitación específica.

1.5. Organización del documento

A continuación se describirá la organización del presente documento explicando brevemente cada uno de los capítulos que lo componen.

Capítulo 2.- En este apartado se presentan las bases teóricas de este trabajo de investigación, cuyo propósito es promover y comprender el desarrollo de este trabajo para lograr estos objetivos.

Capítulo 3.- En esta sección se revisará la tecnología existente y presentará brevemente los trabajos más relevantes para resolver el problema y el algoritmo para resolver el problema.

Capítulo 4.- En esta parte se propone una metodología de trabajo la cual se utilizó para resolver el problema de esta investigación consta de tres partes, 1) la estrada de datos del problema; 2) la forma en como se desarrollaron las tarjetas de desarrollo y 3) la caracterización de las tarjetas de desarrollo.

Capítulo 5.- Esta sección está dedicada a presentar la validación y experimentación realizada para evaluar la metodológia presentada en el capítulo 4.

Capítulo 6.- En este apartado se explicarán en detalle las conclusiones de este proyecto de investigación y el trabajo de seguimiento que se realizará tras este trabajo de investigación.

Capítulo 2

Marco Teórico

En este capitulo se introducen los conceptos relacionados con los problemas de ruteo, los algoritmos que resuelven el problema propuesto, las tarjetas de desarrollo que se estudiaron, los sensores y actuadores utilizados y la caracterización que ayuda a la metodología del problema.

2.1. Problemas de ruteo

Muchos de los problemas combinatorios son computacionalmente intratables, en el sentido de que requieren una cantidad alta de recursos principalmente de tiempo del procesador. El problema de ruteo es uno de los más comunes en la optimización de operaciones logísticas y uno de los más estudiados; plantea la búsqueda de la solución óptima con diferentes restricciones tales como: número de vehículos, su capacidad, lugares de destino (clientes) y demanda de los clientes, entre otras. Una formulación de éste tipo puede incluir un amplio número de variables y diversos parámetros (Bibiana y cols., 2011; Olivera, 2004).

De la formulación propuesta por Flood, nacen variaciones como el TPS generalizado en 1959 con Dantzing y Ramser; trabajo en el cual se modela el despacho de combustible a través de una flota de camiones a diferentes estaciones de servicio, desde una terminal. Este trabajo se convierte en la base para un desarrollo posterior de otras formulaciones que van incrementando el número de variables y restricciones.

Se encuentra la primera referencia del TSP múltiple o m-TSP en 1960 con Miller, Tucker y Zemlin. Este problema es una generalización del TSP en la cual se tiene un depósito y m vehículos, es decir m agentes viajeros. El objetivo planteado es construir exactamente m rutas, una para cada vehículo, de modo que cada cliente sea visitado una vez por uno de los vehículos. Cada ruta debe comenzar y finalizar en el depósito y puede contener a lo sumo p clientes. En el problema m-TSP a cada cliente se le asocia una demanda y cada vehículo cuenta con cierta capacidad, razón por la que se concluye que el problema del agente viajero da origen al problema de ruteo (Yellow, 1970).

Las características de los clientes, depósitos y vehículos, así como diferentes restricciones operativas sobre las rutas y los nodos de inicio y fin, dan lugar a diferentes variantes del problema (Olivera, 2004). Este problema ha despertado ínteres práctico y academico debido a que constituye un problema de optimización combinatoria que no se puede resolver en tiempo polinomial por lo cual pertenece a la clase NP-Hard.

2.2. Métodos de solución para problemas de ruteo

Los algoritmos de solución para el problema propuesto pueden ser agrupados por dos grandes categorías: algoritmos deterministas y no deterministas, ambas categorias son explicadas a continaución.

2.2.1. Algoritmos determinstas

Los algoritmos deterministas exploran todo el espacio de soluciones de un problema, y siempre encuentra el mejor camino posible y han mostrado resultados satisfactorios para la solución de los problemas de ruteo. Se denominan algoritmos deterministas porque solo pueden encontrar una única solución para cada instancia de un problema, independientemente de la frecuencia con la que se ejecute el algoritmo la solución encontrada es siempre la misma, esto implica que el coste de ejecución puede elevarse en gran medida en comparación con otros métodos que van escogiendo la mejor ruta (Azi, Gendreau, y Potvin, 2010).

Algoritmo Dijkstra

El algoritmo de Dijkstra es un algoritmo eficiente de complejidad $O(n^2)$ (donde n es el número de vértices) que sirve para encontrar el camino de coste mínimo desde un nodo origen a un nodo final, con la desventaja de tener que trazar los caminos de coste mínimo a todos los nodos del grafo y, una vez hecho esto, comprueba cúal es el coste del camino que une al nodo inicial con el nodo final, por esto es clasificado como un algoritmo determinista ya que sólo puede encontrar una única solución para un problema (Deng, Chen, Zhang, y Mahadevan, 2012; Jasika y cols., 2012; Salas, 2008).

Sea $G = (V, E)$ un digrafo ponderado con pesos positivos de n vértices. Supongamos que a y z son dos vértices en V , de modo que $z = a$ y existe al menos un camino de a a z. Nuestro principal objetivo consiste en hallar un camino $\underline{a}\ z$ de coste mínimo. El algoritmo inicia en el vértice a y construye un camino de coste mínimo $a = u_0, u_1, \ldots, u_{m-1}, u_m$ tal que $a\ \_u_i$ es un camino de coste mínimo para cada $i = 0, 1, \ldots, m$.

Algorithm 1: Algoritmo Dijkstra

Entrada: graph G; vertices set V; edges set E;
Salida : minimum path covert I

```
1  Initialize a temporal variable d =Ø, pr=Øand S =Ø
2  S = S + {u}
3  D[v] = C[u, v]
4  P[v] = u
5  while (V —S) = Ø do
6  |  foreach v ∈ (V – S) do
7  |  |  D[v] = min(D[v], D[u] + C[u, v])
8  |  |  if min(D[v], D[u] + C[u, v]) == D[u] + C[u, v] then
9  |  |  |  P[v] = u
10 |  |  end
11 |  end
12 end
13 return I
```

Inicialmente S contendra el vertice inicial.y se agregan al vector de costos el costo minimo del nodi inicial. Mientras existan vértices para los cuales no se ha determinado el camino mínimo. Elejimos un vértice que sea el mínimo. Se agrega al conjunto ya que tiene el camino mas corto. Al final se regresa el camino más corto.

2.2.2. Algoritmos no determinstas

Los algoritmos no deterministas están basados en poblaciones de soluciones. A diferencia de los métodos clásicos de mejora basados en seguimiento de trayectorias, en cada iteración del algoritmo no se tiene una única solución sino un conjunto de éstas (Blue et al, 2003).

Estos métodos se basan en generar, seleccionar, combinar y reemplazar un conjunto de soluciones. Dado que mantienen y manipulan un conjunto en lugar de una única solución a lo largo de todo el proceso de búsqueda suelen presentar tiempos de computación sensiblemente más altos que los de otros metaheurísticos. Este hecho se puede ver agravado porque la "convergencia" de la población requiera de un gran número de iteraciones.

Algoritmo evolutivos

Los algoritmos evolutivos son métodos sistemáticos para la resolución del problema de búsqueda y optimización. Éstos tratan de hallar $(x_i, \ldots, x_n)$ tal que $F(x_i, \ldots, x_n)$ sea máximo. En un algoritmo evolutivo, tras parametrizar el problema en una serie de variables, $(x_i, \ldots, x_n)$ se codifican en un cromosoma. Un cromosoma se puede considerar como el contenedor de los valores de las variables. Las soluciones codificadas en

un cromosoma compiten para ver cuál es la mejor solución (Eiben, Smith, y cols., 2003).

Los algoritmo evolutivos enfatizan los nexos de comportamiento entre padres e hijos, en vez de buscar emular operadores genéticos específicos. Estos se basan principalmente en generar una población inicial, después a esta se le aplica una mutación, se calcula la aptitud de cada hijo y se usa un proceso de selección mediante torneo para determinar cuáles serán las soluciones que se retendrán. A continuación se observa el pseudocódigo de un algoritmo evolutivo (Coello, 2016).

Algorithm 2: Pseudocodigo general de un algoritmo evolutivo

Entrada: número de generaciones *g*, población *p*
Salida : best solution *Xbest*

```
1  INITIALISE population whit ramdom candidates solutions;
2  EVALUATE each candidate;
3  foreach TERMINATION CONDITION is satisfied do
4  |   SELECT parents;
5  |   RECOMBINE paire of parents;
6  |   MUTATE the rsulting offspring;
7  |   EVALUATE new candidates;
8  |   SELECT individuals for the next generation;
9  end
10 return best solution)
```

Los algoritmos evolutivos presentan ciertas ventajas en la resolución de problemas de optimización (E. Goldberg, 1989).

- Operan sobre una población (o conjunto de soluciones) lo que evita que la búsqueda se quede atascada en óptimos locales.
- No requieren conocimiento previo sobre el problema a resolverse.
- Pueden combinarse con otras técnicas de búsqueda para mejorar su desempeño.
- Permiten su paralelización de forma sencilla. Son conceptualmente fáciles de implementar y usarse.
- Utilizan operadores probabilísticos, en comparación con las técnicas tradicionales que utilizan operadores determinísticos.
- Generalmente pueden auto-adaptar sus parámetros

2.2.3. Metaheurísticas

El algoritmo metaheurístico es un algoritmo heurístico general, que combina diferentes algoritmos heurísticos definidos según el problema. Estos algoritmos se basan en la búsqueda ambiental y utilizan estrategias aleatorias inteligentes para evitar caer en óptimos locales y mejorar las soluciones obtenidas.

Las metaheurísticas se distinguen por realizar un proceso de búsqueda para encontrar soluciones de calidad aceptable mediante el uso de operadores independientes del dominio, que van modificando las soluciones intermedias guiados por la conveniencia de su función objetivo. Estos algoritmos se basan en la búsqueda de entornos y utilizan estrategias aleatorias inteligentes para evitar que caigan en el óptimo local y mejorar las soluciones obtenidas. Dentro de estos algoritmos se encuentra el Recocido Simulado, Redes Neuronales, Algoritmos Genéticos, Colonia de Hormigas (Toth y Vigo, 2002; Holland, 1975).

Por sus características, este tipo de algoritmo ha sido ampliamente utilizado por la comunidad científica para estudiar problemas complejos en el mundo real. Sin embargo, hasta el momento no se ha encontrado ningún algoritmo que supere a otros algoritmos en todas las situaciones, en esta situación surge un nuevo problema: la selección del algoritmo. Los trabajos relacionados con este tema intentan analizar el comportamiento de algoritmos metaheurísticos con el fin de determinar algoritmos de alto rendimiento. En este trabajo se utilizó un algoritmo genético y una colina de hormigas por lo cual se muestra su descripción acontinuación.

Algoritmo genético

Los algoritmos genéticos GA fueron desarrollados por (Holland, 1975) a principios de los 1960s, motivado por resolver problemas de aprendizaje de máquina. Se basan en la recombinación de soluciones candidatas de una población, que evoluciona por medio de mecanismos genéticos como lo es la selección, cruza y mutación de los individuos de la población. Debido a que para un problema especifico se pueden obtener diferentes soluciones, es clasificado como un algortimo no determinista.

Cuando el algoritmo genético resuelve problemas difíciles, parte de sus buenos resultados depende del equilibrio entre la exploración eficaz y la utilización eficaz. La exploración se refiere a la capacidad de mostrar diferentes partes del espacio de búsqueda en el algoritmo general, mientras que el desarrollo se refiere a la capacidad de ajustar y combinar soluciones subóptimas.

La exploración es muy útil, puede evitar caer en el estado óptimo local y, una vez que esté cerca del valor óptimo global, puede utilizar el desarrollo para obtener el valor óptimo global. En la etapa inicial de la búsqueda, el algoritmo genético debe mostrar una gran diversidad y, finalmente, la diversidad debe reducirse para encontrar una solución (Coello, 2016).

Para comenzar se genera una población inicial a partir de la que se trabaja. Esta

población se suele inicializar de una manera aleatoria, aunque existen implementaciones en las que se obtiene mediante otros métodos como una búsqueda local o cualquier otro tipo de algoritmo si se pretende que la búsqueda se inicie en una determinada dirección del espacio de soluciones. Cuando se han obtenido estos individuos iniciales, se repite el proceso que se explica a continuación tantas veces como indique la condición de parada, que puede ser un numero máximo de generaciones, la convergencia de la población, etc.

El proceso primero evalúa la población y selecciona individuos que interferirán con la formación de la próxima generación. Luego aplique operadores de cruce y mutación y obtenga una nueva población, a partir de la cual se repiten los mismos pasos para avanzar en el proceso de búsqueda (Coello, 2016).

Algorithm 3: Algoritmo genetico

Entrada: número de generaciones *g*, población *p*, porcentaje de cruza *c*, porcentaje de mutación *m*,
Salida : best solution *Xbest*

```
1  Generar_poblacion_inicial()
2  Calcular_fitness()
3  while fin_paro do
4      foreach i ∈ p do
5          Seleccion(i)
6          Cruza(i, c)
7          Mutacion(i, m)
8          Calcular_fitness()
9          Agregar_mejores_individuos()
10     end
11 end
12 return xBest= seleccionarMejor(p)
```

Operador de selección.

El operador de selección determina los mejores individuos de la población actual y los usa para generar la siguiente población. La selección debe garantizar que el mejor individuo (el mejor valor de aptitud) sea más probable que sea seleccionado como padre en el siguiente cruce. Esta posibilidad habilita soluciones que no son soluciones óptimas para informar a la nueva generación.

La posibilidad de elección debe encontrar un equilibrio entre elitismo y exploración. Si la probabilidad de elegir la mejor opción es muy alta, existe el peligro de que los individuos de la población inicial con un estado físico superior al promedio (que representa el óptimo local en lugar del global) se reproduzcan en exceso, lo que resultará en una pérdida de diversidad y un exceso de integración temprana. De lo contrario, la proporción de selección es demasiado baja para provocar búsquedas aleatorias.

Selección por torneo:

Para cada iteración simple se toma cierto número de individuos, llamado también ta-

maño de torneo, y se selecciona para la siguiente generación uno de los individuos del conjunto. Esta operación se repite tantas veces como individuos haya en una población, puesto que en cada selección producimos un solo individuo de la nueva población.

Es evidente que cuanto mayor sea el número de individuos que se hacen competir cada vez, mayor es la presión de la selección para este método. Esto se debe a que el individuo más apto compite contra individuos con peor aptitud de modo que, conforme crece el tamaño de torneo, la probabilidad de tomar una decisión incorrecta también crece (proporcionalmente).

Algorithm 4: Operador de selección por torneo

```
   Entrada: población p
   Salida  : población p_i
 1 Calcular_fitness()
 2 while fin_paro do
 3 |   p_n = barajear_individuos(p)
 4 |   foreach i ∈ p do
 5 |   |   p_p1 = seleccionar_un_individuo(p_n)
 6 |   |   p_p2 = seleccionar_un_individuo(p_n)
 7 |   end
 8 |   p_i = agregar_parejas_individuos(p_p1, p_p1)
 9 end
10 return p_i
```

Operador de cruza.

El operador de cruce implica la selección aleatoria de dos personas de la población para intercambiar sus segmentos de código, produciendo así una "descendencia.ªrtificial cuya descendencia es una combinación de sus padres. Este proceso está diseñado para imitar el proceso de recombinación similar que ocurre en los cromosomas durante la reproducción sexual. Las formas comunes de intercambio incluyen intercambios de puntos, es decir, un punto de intercambio se establece en una ubicación aleatoria en los genomas de dos personas.

Una persona aporta todos los códigos anteriores al punto y la otra aporta todos los códigos en ese punto. El punto produce una progenie y apunta a un cruce uniforme, en el que el valor de una posición dada en el genoma de la progenie corresponde al valor en la posición genómica de un progenitor u otro par de progenitores del valor en esa posición en el genoma, seleccionado 50 % posibilidad.

Algorithm 5: Operador de cruza

Entrada: población p
Salida : población p_i

```
1  Calcular_fitness()
2  while fin_paro do
3      foreach i ∈ p do
4          padre1 = obtener_padre(p)
5          padre2 = obtener_padre(p)
6          hijo1 = obtenerindividuo(padre_1, padre_2)
7          hijo2 = obtenerindividuo(padre_1, padre_2)
8          calcularfitness(hijo_1)
9          calcularfitness(hijo_2)
10         ganadores = verificarfitnes(hijo_1, hijo_2, padre_1, padre_2)
11         pi = agregar_mejores(ganadores)
12     end
13 end
14 return pi
```

Operador de mutación.

El operador de mutación es responsable de realizar pequeños cambios en el código con muy poca probabilidad. Se utiliza para reconstruir la diversidad que puede perderse por sucesivas aplicaciones de operadores de selección y cruce. por lo general Se considera un operador auxiliar, pero esto puede ser un error, porque en ocasiones es muy útil hacer algunas pequeñas modificaciones en la implementación para obtener importantes mejoras en el rendimiento del AG.

Mutación por flip

La forma más común de hacer mutaciones es seleccionar un gen de un individuo y luego cambiar su alelo por otro alelo. La probabilidad de estos cambios suele ser baja (generalmente 1 por persona) para prevenir la AG realice una búsqueda aleatoria. Sin embargo, en ocasiones puede ser necesario incrementar esta posibilidad para recuperar la diversidad de la población.

Algorithm 6: Operador de mutación

Entrada: población p
Salida : población p_i

```
1  Calcular_fitness()
2  while fin_paro do
3  |  individuos = probabilidad_mutacion()
4  |  foreach i ∈ p do
5  |  |  individuo = individuos(p)
6  |  |  nuevo_individuo = aplicar_mutacion(individuo)
7  |  |  p_i = agregar_nuevo_individuo(nuevo_individuo)
8  |  end
9  end
10 return p_i
```

Algoritmo Colonia de hormigas

El sistema de hormigas se inspira en la estrategia de la colonia de hormigas para encontrar comida. Cuando una hormiga encuentra el camino hacia una fuente de alimento, deposita una sustancia llamada feromona en el camino. La cantidad de feromona depositada depende de la longitud del camino y de la calidad del alimento encontrado (Barcos, Rodríguez, Álvarez, y Robusté, 2002; Hincapié, Porras, y Gallego, 2004).

Si una hormiga no detecta la presencia de feromonas, se moverá al azar. Sin embargo, si se detectan las sustancias mencionadas anteriormente, lo más probable es que decida moverse a lo largo de un mayor número de trayectorias, lo que dará como resultado un aumento de la feromona depositada en el área. De este proceso surgió un comportamiento llamado autocatálisis: cuantas más hormigas siguen un camino específico, más atractivas son.

Una forma en que las hormigas explotan la feromona para encontrar el camino más corto entre dos puntos trabaja de la siguiente manera, primero las hormigas llegan a un punto en el cual deben decidirse por uno de los dos caminos a seguir; despúes se realiza una elección de camino, la que puede ser aleatoria al haber bajos niveles de feromona o guiada al haber una diferencia notable entre la cantidad de feromona de cada camino (Guerra, Valadez, Soberanes, y de Montserrat Ortiz-Aguilar, s.f.).

Por consiguiente, dado que la velocidad de una hormiga se considera aproximadamente constante, es claro decir que las hormigas que eligieron el camino más corto se demoren menos que las otras en llegar hasta el otro extremo, lo cual conduce a una mayor acumulación de feromona en el camino. Al final, la feromona acumulada en mayor cantidad en el camino más corto y más circulado guía a las hormigas a la fuente de alimento de la forma más rápida, donde el camino menos transitado pierde la feromona depositada en él a causa de la evaporación.

Reglas de aprendizaje para el algoritmo colonia de hormigas

Se describe las dos reglas de aprendizaje clásicas, originalmente propuestas por Dorigo en el algoritmo ACS. La primera regla es llamada de transición de estado; con esta regla el siguiente nodo a ser visitado es seleccionado. La regla de transición usa dos estrategias: explotación y exploración. Ecuaciones 2.1 y 2.2.

$$s = \begin{cases} \operatorname{argmax}_{p \in p_i} \{x\} * x_r & \text{if } q \leq q_0 (explotacin) \\ S & \text{de otra forma (exploración)} \end{cases} \quad (2.1)$$

$$s = p_{r,u,t} = \frac{x + x_r}{\sum_{\forall p \in p_i} x + x_r} \quad (2.2)$$

Donde r es el nodo actual donde se localiza la hormiga k, x_r es el conjunto de nodos visitados por la hormiga k, β es el parámetro que determina la importancia relativa entre la feromona y el DDC con la función de importancia de la distancia, p_i es la lista de nodos disponibles, q es un número aleatorio entre [0,1] y q_0 determina la importancia relativa de la exploración contra la explotación.

En caso que $q \leq q0$, la estrategia de explotación es seleccionada: ésta selecciona un nodo que provee una gran cantidad de feromona y una mejor conectividad con un número pequeño de saltos hacia un recurso. De otra forma la estrategia de exploración es seleccionada a través de la Ecuación 2.2.

La segunda regla es llamada de actualización; con esta regla la hormiga actualiza los nodos en la ruta viajada. La regla de actualización usa dos estrategias: actualización local y global de la feromona. Estas estrategias son usadas para retroalimentar el sistema sobre rutas exitosas.

Las reglas de actualización de la colonia de hormigas están compuestas de actualización local y global. En consecuencia la regla de actualización modificada en este trabajo es dada en las Ecuaciones 2.3 y 2.4. La regla de actualización local de algoritmo esta formulada por:

$$\tau_{r,s,t} = (1 - p) * \tau_{r,s,t} + p * \tau_0 \quad (2.3)$$

donde r es el nodo actual en el cual la hormiga k esta localizada, s es el nodo vecino actual al cual la hormiga se va a mover, t es el recurso buscado, τ es la tabla de feromonas usadas por la hormiga para hacer la actualización local, τ_0 es el valor de inicialización de la tabla de feromonas y p es el factor de evaporación local de la feromona.

Cada vez que una hormiga decide moverse hacia un nodo mediante la regla de selección, algo de feromona es depositado en cada nodo que ha sido visitado para establecer una ruta hacia los nodos mas frecuentemente visitados. Por otro lado, la regla de actualización global modificada, dada por la Ecuación 2.1, es calculada cada vez que un recurso es encontrado y es aplicada en cada uno de los nodos que pertenecen a la ruta a través de la cual se descubrieron los recursos

$$\tau_{r,s,t} = (1 - \alpha) * \tau_{r,s,t} \qquad \forall r, s, t \in k \tag{2.4}$$

donde α es el factor de evaporación global de la feromona.La cantidad de feromona depositada depende de la calidad de la solución obtenida, la cantidad de recursos encontrados y la importancia del tiempo de vida de la hormiga.

Para este problema, los modos de operación de estos algoritmos se resumen a continuación: el algoritmo se inicializa colocando una hormiga en cada nodo. Para la construcción de carreteras, utilice reglas de probabilidad si El nodo ya ha sido visitado y en el caso contrario es cero. Es más probable que las hormigas visiten el nodo. En cada arco, si la solución obtenida es menor que el límite predeterminado, la "feromona"se actualiza y finaliza, de lo contrario se recalculará la probabilidad y la hormiga seguirá construyendo la solución (Núñez y Dietrich, s.f.; Coronel, Santacruz, Ríos, y Cegla, 2006).

Algorithm 7: Algoritmo colonia de hormigas

Entrada: número de hormigas *N*; trazo inicial de feromonas *F*; estructura de vecinada de hormigas *C*; estructura de memoria colectiva de hormigas *M* ; estructura de probabilidad para cada movimiento de una hormiga *P*

Salida : best solution *Xbest*

```
1  while termination condition not met do
2      ConstructSolutionsbyAnts()
3      foreach i= 1 to N do
4          M = ActualizarMemoria(M)
5          while not Solucionconstruida do
6              C = MovimientosPosibles(pos)
7              P = EvaluarMovimientos(C,M)
8              pos= MovimientoHormiga(P)
9              F = DepositarFeromonas(pos)
10             x := AnnadirComponente(pos)
11         end
12         x = ActualizarSoluciones(x)
13     end
14     F = EvaporacionFeromonas(x,F)
15     F = AccionesDemonio(x,F)
16 end
17 return xBest= seleccionarMejor(x)
```

La definición de los rastros de feromonas es una tarea crucial para un problema dado, tanto que una mala elección puede conducir a que las soluciones que se obtengan con ACO sean de baja calidad.

Como en cualquier metaheurística, ACO establece un compromiso entre la diversi-

ficación y la intensificación del procedimiento de búsqueda. Concretamente, se realiza a través de la gestión del rastro de feromonas (tanto en el depósito como en la evaporación). Dicho rastro describe una función de distribución de probabilidad sobre el espacio de búsqueda, determinando qué partes de dicho espacio de soluciones se deben muestrear más intensamente.

2.3. Sensores

El sensor se basa en el hecho de que el sensor no solo cambia el rango de la variable física medida, sino que la salida del sensor también es un dato útil para el sistema de medición. Por lo tanto, un sensor se define como un dispositivo de entrada que proporciona una salida manipulable de una variable física medible. Por tanto, el sensor puede ser un dispositivo de entrada, porque este último siempre será un intermediario entre la variable física y el sistema de medida (Garcia, 2015; Latha, Murthy, y Kumar, 2016).

Los sensores se pueden clasificar de muchas formas, para el uso de este trabajo se puede hacer por el tipo de variable a medir. Esta clasificación suele ser común porque los sensores se pueden usar para medir diferentes variables físicas. Sin embargo, su principio de funcionamiento es el mismo y solo depende del tipo de configuración donde se coloque y cómo se interprete la señal proveniente de él (Pawar, 2014).

Tabla 2.1: Clasificación de los sensores.

Clasificación de los sensores según la variable física a medir.	De posición, velocidad y aceleración. De nivel y proximidad. De humedad y temperatura. De fuerza y deformación. De flujo y presión. De color, luz y visión. De gas y pH. Biométricos. De corriente.

2.3.1. Sensor HC-SR04

El sensor HC-SR04 es un sensor de distancia ultrasónico. Es un módulo que incorpora un par de transductores de ultrasonido que se utilizan de manera conjunta para determinar la distancia del sensor con un objeto colocado enfrente de este. Un transductor emite una ráfaga de ultrasonido y el otro capta el rebote de dicha onda (Latha y cols., 2016). El tiempo que tarda la onda sonora en ir y regresar a un objeto puede utilizarse para conocer la distancia que existe entre el origen del sonido y el objeto.

Figura 2.1: Sensor HC-SR04

Existen 2 tipos de módulos para medición de distancia por ultrasonido que se utilizan en robótica y aplicaciones similares: a)Interfaz mediante pulso de eco: poseen un pin de disparo y otro de eco. La medición se obtiene mediante un pulso digital. b) Interfaz serial (I2C o UART): entregan su medición en formato digital a través de una interfaz serial (I2C o UART).

2.3.2. Sensor de humedad y temperatura

DHT11 es un sensor digital de temperatura y humedad, que integra un sensor de humedad capacitivo y un termistor para medir el aire circundante, y muestra datos a través de una señal digital en el pin de datos (sin salida analógica) (Ferdoush y Li, 2014). El único inconveniente del sensor puede ser que solo puede obtener nuevos datos cada 2 segundos.

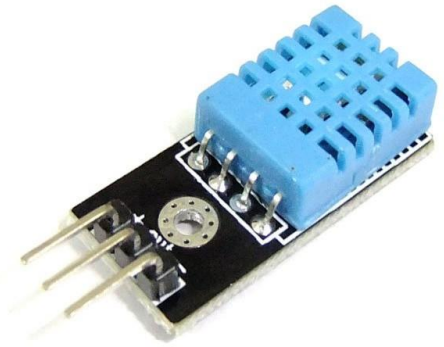

Figura 2.2: Sensor temperatura y humedad

Cada sensor se calibra en fábrica para obtener coeficientes de calibración registrados en su memoria OTP, lo que garantiza una alta estabilidad y confiabilidad a largo plazo. El protocolo de comunicación entre el sensor y el microcontrolador utiliza un solo hilo o cable. La longitud máxima recomendada del cable es de 20 m, y los cables blindados son los mejores. Proteja el sensor de la luz solar directa (radiación ultravioleta).

2.3.3. Sensor detector de fuego

Como medida de seguridad, es muy importante detectar la actividad del fuego (llama). En algunos casos, debemos recordar si hay un incendio (en un incendio), o en otros casos, como en un sistema de calentamiento de llama, si la llama está funcionando.

El módulo del sensor fotoeléctrico es sensible a la luz infrarroja (760-1100nm) y está polarizado y permite que la corriente pase cuando hay un incendio, por lo que el módulo del sensor de llama puede detectar si hay un incendio. La corriente fluye a través de la resistencia y produce una caída de voltaje, que se pasa a la salida analógica y se utiliza como entrada de un amplificador operacional comparador que proporciona una salida digital. El nivel de sensibilidad se puede ajustar con un potenciómetro (Khalaf, Abdulsahib, y Zghair, 2019).

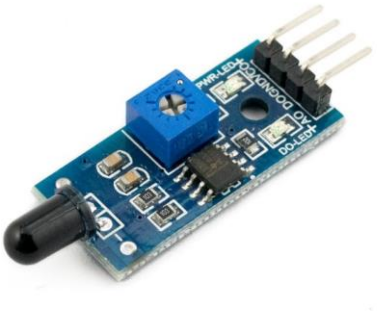

Figura 2.3: Sensor detector de fuego

2.4. Actuadores

Un actuador es un dispositivo que puede generar fuerza a través de la conversión de energía, que ejerce una determinada posición, velocidad o cambio de estado sobre un elemento mecánico.

Los actuadores se dividen en varias categorías, pero a los efectos de este trabajo, se utiliza el tipo de energía utilizada: actuadores eléctricos.

Los actuadores eléctricos convierten la energía eléctrica en energía mecánica, ya sea giratoria o lineal, son los más utilizados porque consisten en una gran cantidad de filamentos conductores (llamados vueltas). Dado que la corriente que circula en un conjunto de vueltas es magnética, provoca un movimiento circular en el eje del actuador y genera esta energía mecánica.

2.4.1. Motor motoreductor

Un motorreductor es una unidad compacta y homogénea formada por un reductor y un motor. Tiene una determinada potencia en HP y tiene una cierta velocidad de operación a la cual gira la flecha de salida, por ejemplo 100 Revoluciones por Minuto (RPM) (Akbar y cols., 2014).

Figura 2.4: Motor motorreductor

El papel principal en un motorreductor lo desempeña el reductor y sus fases, los pares. Estas características transmiten la fuerza del motor del eje de entrada al eje de salida. Por lo tanto el reductor funciona como un variador de velocidad y par.

2.4.2. Motor paso a paso

Es un motor paso a paso unipolar que convierte pulsos eléctricos en movimientos mecánicos exactos girando en ángulos fijos cuando se le aplican impulsos eléctricos en la secuencia correcta.

Un motor paso a paso está diseñado para girar un paso cada vez que alimentas una de las fases. Para el paso full step se excitan un par de bobinas, después se excita otro par teniendo en común una bobina que esta excitada del par anterior (Akbar y cols.,

2014). La posición del movimiento del imán es de manera diagonal.

Figura 2.5: Motor paso a paso

Por otro lado, el paso half step primero se excita una bobina, luego dos a la vez, luego la siguiente y así hasta completar la secuencia. En la Figura se muestra el diagrama de secuencias descritas anteriormente.

2.4.3. Controlador L29

Permite encender y controlar dos motores de corriente continua, variando la dirección como la velocidad de giro. Consiste en dos puentes-H, uno para la salida A y otro para la salida B.

Un puente-H es un componente ampliamente utilizado en electrónica para alimentar una carga de forma que podemos invertir el sentido de la corriente que le atraviesa.

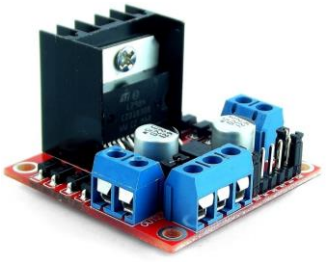

Figura 2.6: Controlador puente H

Internamente un puente-H es una formación de 4 transistores, conectados entre Vcc y GND; donde Vcc es la terminal de corriente y GND es la terminal de tierra; con la carga a alimentar entre ellos. Actuando sobre los 4 transistores, activando los transistores opuestos en diagonal de cada rama, podemos variar el sentido en el que la corriente

atraviesa la carga.

2.5. Tarjetas de desarrollo

Las tarjetas de desarrollo son placas electrónicas que son útiles al realizar cualquier tipo de sistemas embebidos, donde cada una, en particular, cuenta con su lenguaje de programación o un sistema operativo. Para estas existen 2 clasificaciones:

1. Placas de desarrollo con microcontroladores.
2. Placas de desarrollo minicomputador.

2.5.1. Arduino

Este tipo de tarjetas están compuesta por algún microcontrolador de alguna empresa tales como Microchip, Atmel, etc. o también de plataformas como ARM, y solo responden a un lenguaje en particular, ejemplo: C, C++, etc. Dentro de las tarjetas mas utilizadas son las de la marca Arduino (Arduino, s.f.).

Arduino es una plataforma electrónica de código abierto basada en hardware y software y una gran comunidad tecnológica que diseña y elabora las placas de desarrollo. Arduino recopila la información obtenida de cualquiera de sus pines de entrada, en los cuales se pueden conectar sensores y actuadores que permiten construir sistemas que perciben realidad y responden con acciones físicas con el entorno.

El microcontrolador de la placa se programa con el “Arduino Programming Lenguaje”, y el “Arduino Development Enviroment”, que engloba todas las funciones básicas de los microcontroladores AVR, lo cual permite una programación más sencilla y aun nivel más elevado (Herrador, 2009). Principales ventajas de la utilización de arduino como placa microcontroladora:

El software de arduino es multiplataforma, este puede ser ejecutado en sistemas operáticos Windows, GNU/Linux, Macintosh OSX. La curva de aprendizaje que genera el entorno de programación de arduino, concibe que el conocimiento y tiempo de aprendizaje del lenguaje de programación de la placa, sea flexible para los usuarios. El software arduino esta publicado como herramientas de código abierto, por tal motivo está disponible la extensión de la aplicación desarrollada por programadores experimentados. Arduino está basado en los microcontroladores ATMEGA8 y ATMEGA168 de Atmel.

Arduino UNO

En la figura se muestra la tarjeta arduino UNO, la cual fue selecionada para este trabajo de investigación. Esta tarjeta es una board con un microcontrolador At-

Tabla 2.2: Modelos de arduino y sus caracteristicas.

Nombre	Microcontrolador	Velocidad (CPU)	EEPROM [K-bytes]	SRAM [K-bytes]	FLASH [K-bytes]
UNO	ATMega328	16 Mhz	1	2	32
Due	AT91SAM3X8E	84 Mhz	-	96	512
Leonardo	ATMega32u4	16 Mhz	1	2.5	32
Mega 2560	ATMega2560	16 Mhz	4	8	256
Mega ADK	ATMega2560	16 Mhz	4	8	256
Micro	ATMega32u4	16 Mhz	1	2.5	32
Mini	ATMega328	16 Mhz	1	2	32
Nano	ATMega328	16 Mhz	0.512	2	16
Ethernet	ATMega328	16 Mhz	1	1	32
Esplora	ATMega32u4	16 Mhz	1	2.5	32
ArduinoBT	ATMega328	16 Mhz	1	2	32
Fio	ATMega328P	8 Mhz	1	2	32
Pro (168)	ATMega168	8 Mhz	0.512	1	16
Pro (328)	ATMega328	16 Mhz	1	2	16
Pro Mini	ATMega168	8 Mhz	0.512	1	32
LiLyPad	ATMega168V	8 Mhz	0.512	1	16
LiLyPad USB	ATMega32u4	8 Mhz	1	2.5	32
LilyPad Simple	ATMega328	8 Mhz	1	2	32
LilyPad SimpleSnap	ATMega328	8 Mhz	1	2	32

mega328. Tiene 14 pines de entrada /salida digital y 6 entrada/salida análogicas.

2.5.2. Raspberry Pi

Es un dispositivo que dispone de las prestaciones básicas de un ordenador concentrados en una sola placa con la intención de mantener un coste bajo, tiene una arquitectura con microprocesador o microcontrolador, memoria RAM (Random Access Memory), memoria ROM (Read-only memory) y entradas/salidas todo en la misma placa, es capaz de soportar un sistema operativo y la mayoría tiene una plataforma ARM (Raspberry, s.f.).

Raspberry Pi es un SBC (de las siglas en inglés Single Board Computer) de tamaño reducido, desarrollada en el Reino Unido por la Fundación Raspberry PI en la Universidad de Cambridge en 2011, con un propósito educacional, empezó su comercialización hasta el año 2012 y pese a su tamaño que es aproximadamente el de una tarjeta de crédito permite realizar grandes cosas. Existen varias versiones de esta placa, en cada versión se han ido actualizando e incorporando componentes más modernos, pero todas comparten algunas características.

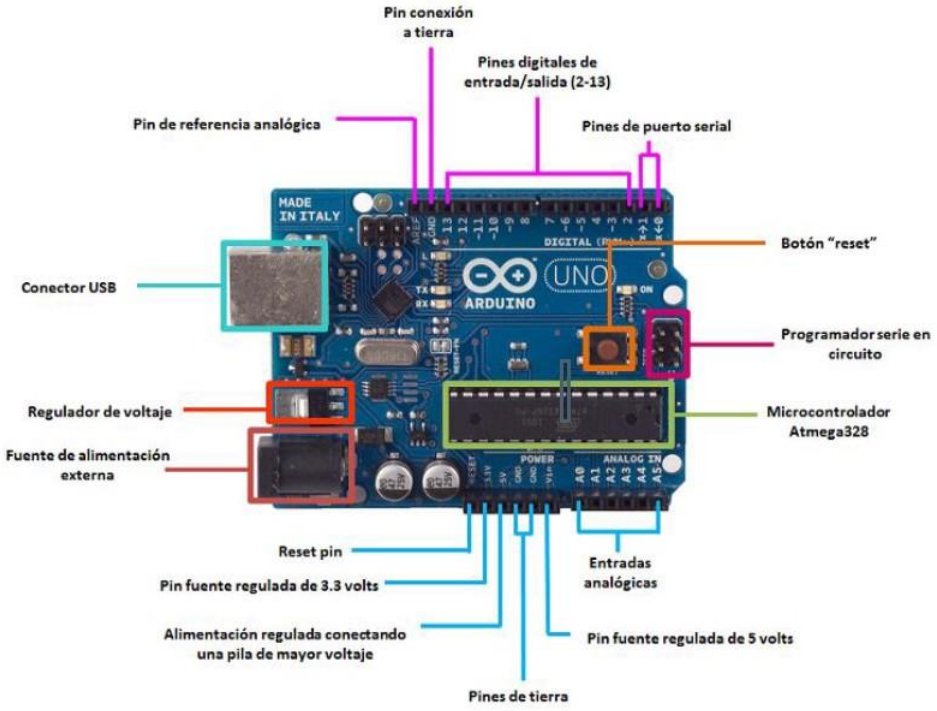

Figura 2.7: Arduino UNO

Principales ventajas de la utilización de raspberry pi como placa microcomputadora(Rodríguez, 2018): Tamaño reducido. La mayoría no rebasa los pocos centímetros y son ideales para integrarlas en sistemas con espacio limitado. Alta eficiencia. Gastan muy poca energía y producen muy poco calor. Fácil aislamiento. Gracias a su reducido tamaño es muy sencillo proteger la placa de la computadora en condiciones ambientales adversas. Bajo costo. Muchas de estas placas tienen un costo accesible y ofrecen buenas prestaciones (Pi, 2015).

Pero también se tienen algunas desventajas al trabajar son estos sistemas. Falta de escalabilidad. A diferencia de las computadoras modulares a las que se les conectan componentes por separado eligiendo cada uno de ellos según los requerimientos, en una computadora modular esto es imposible, pues los componentes vienen soldados en la misma placa. Difícil reparación. Ya que no son modulares, si alguno de los componentes presenta fallos, remplazarlo es una tarea muy difícil y se opta por remplazar la placa completa.

Raspberry Pi

En la figura se muestra la tarjeta Rasperry pi, la cual fue seleccionada para este trabajo de investigación.

Tabla 2.3: Modelos de raspberry pi y sus caracteristicas.

Nombre	Velocidad	Processor Chipset	RAM	GPIO
RPi Model A+	Single Core 700 MHz	Broadcom BCM2835 32Bit	256MB SDRAM	40
RPi Model B+	Single Core 700 MHz	Broadcom BCM2835 32Bit	512MB SDRAM	40
RPi 2 Model B	QUAD Core 900 MHz	Broadcom BCM2836 32Bit	1GB SDRAM	40
RPi 3 Model B	QUAD Core 1.2 GHz	Broadcom BCM2837 64Bit	1GB SDRAM	40
RPi 3 Model B+	QUAD Core 1.4 GHz	Broadcom BCM2837B0 64Bit	1GB SDRAM	40
RPi 4	QUAD Core 1.5 GHz	ARM Cortex-A72 (4 nucleos) 1.5 GHz	1GB/2GB/4GB	40

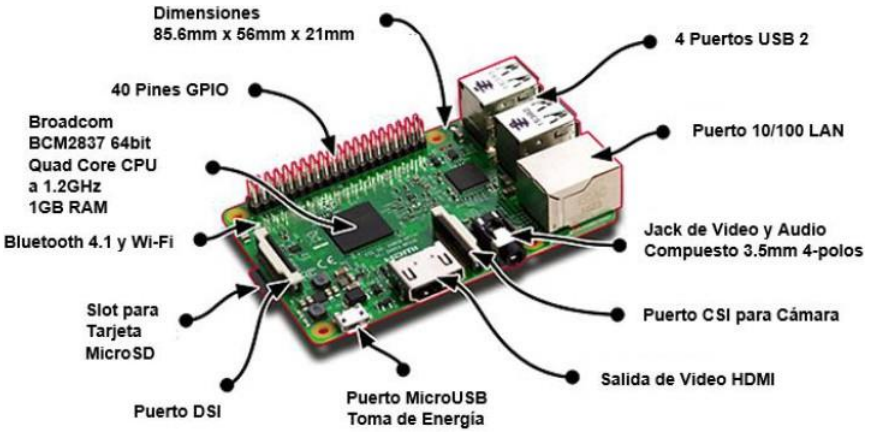

Figura 2.8: Raspberry Pi

2.6. Caracterización

Las características son todos los aspectos que definen un objeto ya sea físicamente o lógicamente incluyendo su comportamiento en el medio en que se desenvuelve. Mientras más se conocen y comprenden las características de un objeto más fácil será trabajar con el. De ahí el concepto de caracterización, que centrado a la computación, puede denominarse como el proceso en el que se identifican todos los factores que intervienen en la resolución de un problema, desde la estructura de las instancias hasta el comportamiento del algoritmo, pasando por su uso y la estructura de las soluciones obtenidas (Quiroz, 2009).

La caracterización del problema y el proceso de solución son una parte esencial del análisis del desempeño del algoritmo, y con ello es posible determinar qué factores afectarán el desempeño del algoritmo. Para realizar un análisis de desempeño de calidad se require de las metricas adecuadas que cuantifiquen las caracteristicas indicadoras del desempeño. En (Barr S., 1995) sugiere que hay tres categorías principales de factores que afectan el rendimiento del algoritmo: *problema, algoritmo y ambiente*.

- ***Factores del problema:*** Son los factores que definen la instancia del problema a resolver: dimensión, distribución de los parámetros, estructura del espacio de soluciones, entre otros.
- ***Factores del algoritmo:*** Incluyen las estrategias heurísticas utilizadas (procesos de construcción de solución inicial y parámetros de búsqueda asociados), códigos de computadoras empleados, configuración de control interno del algoritmo y comportamiento en la ejecución, entre otros.
- ***Factores del ambiente:*** Estos factores se refieren al ambiente físico en el que serán ejecutados los algoritmos como los son el software y hardware. Así como también aquellos relacionados con el programador.

Para este trabajo en los factores del problema se describiran con las caracterisitcas de las intancias, las cuales son seleccionadas por la cantidad de nodos que contiene, además las estructuras del espacio de soluciones son dadas dependendiendo del caso de la tarjeta de desarrollo.

En relación de los factores del algoritmo, se caracterixan mediente los algoritmos que son desarrollados, analizando la creación de soluciones iniciales y como se va desarrolando la población de soluciones, ade más de los indicadores de tiempo de ejecución y memoria utilizada.

Para el indicador *tiempo* representa el tiempo utilizado en cada tarjeta de desarrollo cuando logra resolver un algoritmo, este esta dado en milisegundos y se toma desde que inicia el proceso de solución, hasta que da un resultado favorable.

El indicador de *memoría* representa la cantidad de memoria utilizada por cada tarjeta de desarrollo la cual es utilizada por estas al resolver los algoritmos, este indicador se expresa en kbytes y se da a la cumulación de memoria cuando inicia la tarjeta hasta que termina por completo el proceso.

Para este trabajo en particular se interesa en los factores del problema y del algoritmo, por lo cual se usarán las metricas o parámetros de estos dos factores para obtener conocimiento del conjunto de datos y de su estructura subyacente incluyedo métodos estadísticos y gráficos que ayuden a obtener un análisis que describe el conjunto de relaciones de los factores bajo estudio.

Capítulo 3

Estado del arte

En la literatura publicada se encuentran diversos casos de estudio enfocados a la salud, finanzas, clasificación de imágenes por mencionar algunos, los cuales han sido implementados usando diversas tarjetas de desarrollo a través de técnicas como redes neuronales, algoritmos genéticos, algoritmos exactos, etc. En la siguiente sección se realizó una búsqueda acerca de como las tarjetas de desarrollo han sido evaluadas con respecto al problema a solucionar.

Proyecto: Controlador difuso-evolutivo para robot seguidor de luz

(Arvizu, Carlos, Liñan, Angel, y Torres Treviño, 2016) propone un controlador difuso-evolutivo, que está compuesto por un sistema de lógica difusa de tipo I y un algoritmo genético con cromosomas de genes enteros y el sistema robótico evolutivo completo consta del controlador difuso-evolutivo y de un prototipo robótico, el cual no requiere el conocimiento previo de una persona experta acerca de la dinámica del sistema, además se plantean experimentos de seguimiento de luz en cuatro escenarios de prueba, incluyendo perturbaciones y variaciones lumínicas, lo que ilustra la adaptabilidad del sistema evolutivo.

El hardware requerido para el sistema es el siguiente:

- Microcontrolador: La tarjeta electrónica arduino Due contiene el código computacional del controlador difuso-evolutivo. Las características más importantes para la selección de esta tarjeta son principalmente, la capacidad de procesamiento y el espacio de memoria disponible para almacenamiento de código y datos.
- Servomotores: El robot tiene dos servomotores, un servomotor controlado por la salida del sistema de lógica difusa y un servomotor de la fuente lumínica para la posición de esta en las etapas de calibración, evaluación y el modo de pruebas del sistema.
- LED de Potencia: Su objetivo es proporcionar la fuente lumínica que sigue el robot.
- Sensores LDR: Se utilizan sensores fotosensibles en el robot para medir la cantidad de luz de su entorno. Los valores de los sensores son las entradas al sistema

de lógica difusa.

Para la experimentación se mide el desempeño a través de los valores de aptitud de las soluciones generadas y el error promedio concluyendo que es necesario hacer ajustes en el software, como calibración inicial, de modo que las mediciones obtenidas sean precisas y el algoritmo genético pueda trabajar para obtener mejores resultados.

Proyecto: raspberry pi como plataforma de algoritmos de machine learning: Reconocimiento de imágenes y datos financieros en streaming

El objetivo del trabajo de Rodríguez (Rodríguez, 2018) es evaluar la implementación de modelos predictivos basados en redes neuronales y técnicas de Machine Learning en dispositivos con capacidad de cómputo relativamente baja para modelos de esta índole.

Para este estudio el dispositivo seleccionado ha sido raspberry pi 3 debido a su potencia de cómputo, además es modular ya que se le puede proporcionar diferentes funcionalidades como el uso de la Pi Camera o conexión de arduinos.

Las aplicaciones que se abordaron son la clasificación de objetos mediante técnica de reconocimiento de imágenes y la predicción de valores futuros de datos financieros en procesos de datos en streaming.

Las métricas de desempeño de raspberry Pi con reconocimiento de imágenes para los algoritmos Red Convolucional (CNN) y el clasificador logístico son el tiempo de respuesta, la precisión de la cámara y del conjunto de prueba.

La metodología seguida para el análisis de datos financieros se basa en el estudio de las series temporales en donde se estudió el comportamiento de Raspberry pi con 5 diferentes modelos cada uno de ellos con diferentes capas (desde 1 capa hasta 5 capas) y se midió el error cuadrático medio de la predicción (MSE) en tiempo real y el tiempo que tarda en dar dicha predicción.

Proyecto: Aplicación de path planning de un robot móvil basado en raspberry pi

En el trabajo de Cabezas (Cabezas, 2017) se realizó un prototipo de procesado de datos y PathPlanning con ayuda del montaje y programación de un vehículo robot mediante el uso de una raspberry pi 3.

Debido a la capacidad de aplicar un algoritmo de planificación de ruta fue seleccinado la tarjeta Raspberry Pi a la cual se le insertó un sensor de control de motores para el manejo de las ruedas y un GPS el cual le da una posición global al dispositivo.

La problemática a resolver la define como dado un terreno en ausencia de obstáculos y completamente llano, se busca localizar un camino (siempre que exista) entre

un origen y un destino seleccionados por el usuario donde el algoritmo utilizado para resolverlo es el algoritmo Yowi.

Las métricas de desempeño de la tarjeta raspberry pi son la validación de los datos mandados a la base de datos, también el tiempo estimado para el cálculo de la mejor ruta a tomar.

Proyecto: Vehículo autónomo para gestión de trayectorias basado en arduino y raspberry pi

En el proyecto de Goméz (Gómez, 2017) se desarrolló un sistema autónomo capaz de desplazarse por el mundo físico desde un lugar de origen hasta un destino y que pueda ser controlado fácilmente por un usuario mediante una aplicación web, capaz de operar con un alto grado de autonomía, con un pequeño aporte de información proveída por un usuario.

Las tarjetas de desarrollo elegidas en el proyecto fueron raspberry pi debido a que tiene la potencia suficiente para este proyecto, por su memoria disponible y su capacidad de almacenamiento. Además, fue utilizado arduino por su facilidad de manejo de sensores y actuadores, además de que su arquitectura es eficaz para este tipo de problemas.

El algoritmo empleado para este proyecto es el algoritmo A*, el cual es un algoritmo para la búsqueda de caminos entre nodos. El tiempo de respuesta de la tarjeta raspberry pi y arduino es la métrica utilizada para la evaluación de las tarjetas.

Prototipo de sistema experto basado en lógica difusa para la monitorización del ruido en espacios educativos.

El siguiente trabajo de (Aracelis, Casco, y Pinzón, 2020) presenta un prototipo de un sistema experto basado en lógica difusa para la monitorización del ruido y un mecanismo de alerta en espacios educativos. El sistema desarrollado plantea la utilización de técnicas de inteligencia artificial para la toma de decisión y mecanismos de salidas que actúan como alertas, concretamente mediante semáforos de estados, indicando el nivel de ruido captado.

Se propone una arquitectura de hardware y software encargados de la captación y procesamiento de los decibeles captados. Para su desarrollo se utilizaron los siguientes materiales: Placa raspberry pi 3B, placa arduino Mega, Sensor de sonido ky_038, Relay, Lámpara de led (verde, amarillo, rojo).

El funcionamiento del sistema comienza en la captación de los decibeles mediante el sensor del sonido que son recibidos por Arduino Mega para después ser transmitidos a la placa central Raspberry Pi, una vez que finaliza el intervalo de captación de datos, se calcula un promedio de los valores registrados. Mediante las reglas de lógica difusa se clasifica el valor promedio, dando como resultado la asignación de un estado

comprendido en verde, amarillo o rojo.

Comentarios finales

Después de hacer la revisión del estado del arte, se muestra un resumen de los trabajos revisados de la literatura concentrado en la figura 3.1 haciendo una comparativa con el trabajo que se desarrolla en este proyecto. Se pude observar que las metricas de medición en la mayoria de los trabajos es el uso del tiempo de ejecución. Tambien que se usan como datos de entrada mas el uso de sensores. Hay una gran variedad de algoritmos implementados, y que es mas frecuente el uso de tarjetas arduino que de raspberry.

Figura 3.1: Tabla de comparación de los trabajos de la literatura.

Autor	*Tarjetas de desarrollo*	*Caso de estudio*	*Método de solución*	*Métricas de evaluación*	*Datos de entrada*	*Sensores y/o actuadores*
Arvizu et al (2016)	Arduino DUE	Inferencia a través de luz ambiental	Algoritmo genético con reglas difusas	Error promedio Valores de aptitud	Condiciones de luz ambiental a través de un sensor LDR	Servomotores LED de potencia Sensor LDR
Rodríguez, G (2018)	Arduino DUE Raspberry Pi 3	Reconocimiento de imágenes.	Redes neuronales	Tiempo de respuesta Precisión de la cámara Pi Error promedio cuadrático	Foto tomada por la cámara pi Test set	Cámara Pi
Cabezas, F. (2017)	Raspberry Pi 3	Encontrar un camino de origen y un nodo destino	Algoritmo Yowi	Tiempo de respuesta	Posición global a través del GPS	GPS controlador de motores
Gómez, F (2017)	Arduino UNO Raspberry Pi 3	Encontrar un camino de origen y un nodo destino	Algoritmo A*	Tiempo de respuesta	A través de una aplicación donde el usuario los proporciona	Servomotores
Aracelis et al. (2020)	Raspberry Pi 3 Arduino Mega	Desarrollo de técnicas de inteligencia artificial para la toma de decisiones	Reglas de lógica difusa	Valor promedio	A través de sensores que captan el ruido del ambiente	Sensor de sonido Replay lampara de Led
Haridas, M. et al. (2017)	Arduino Mega	Desarrollar un sistema que permite encontrar el costo mínimo del árbol de expansión del enlace a diferentes nodos en una red eléctrica	Algoritmo Dijkstra	Costo mínimo del árbol de expansión Tiempo de ejecución	Conjunto de instancias tomadas de la literatura	Sensor de luz Ultrasonido
Para este trabajo	Raspberry Pi 3 Arduino UNO	Solucionar un problema de ruteo propuesto a través de metaheurísticas	Metaheurísticas	Tiempo de ejecución Consumo de memoria Rangos de dominio entre tarjetas	Conjunto de instancias tomadas de la literatura Sensores	Sensor de temperatura Sensor de flama Ultrasonido

En la figura 3.2 se destacan las tarjetas de desarrollo utilizadas por los diferentes autores, también se indica el caso de estudio a resolver, así como los métodos de solución implementados para la solución del problema y las métricas de evaluación de las tarjetas de desarrollo empleadas comparandolas con las que utilizan en este proyecto de tesis.

Figura 3.2: Estado del arte en ánalisis de desempeño de tarjetas de desarrollo

Autor	Tarjetas de desarrollo		Método de solución			Métricas de evaluación				Datos de entrada	
	Arduino	Raspberry Pi	Algoritmo exacto	Algoritmo metaheurístico	Reglas de lógica difusa	Tiempo de respuesta	Valor promedio	Rangos de dominio	Consumo de memoria	Sensores	Instancias de la literatura
ARVIZU ET AL (2016)	✓			✓			✓			✓	
RODRÍGUEZ, G (2018)	✓	✓		✓		✓				✓	
CABEZAS, F. (2017)		✓	✓			✓				✓	
GÓMEZ, F (2017)	✓	✓	✓				✓				✓
ARACELIS ET AL. (2020)	✓	✓			✓		✓			✓	
HARIDAS, M. ET AL. (2017)	✓		✓			✓					✓
PARA ESTE TRABAJO	✓	✓		✓		✓		✓	✓	✓	✓

Capítulo 4

Metodología de solución

En esta sección se presenta la metodología propuesta para el cumplimiento de los objetivos de este trabajo de tesis, el cual se conforma por un conjunto de cuatro etapas que permitirán conocer el desempeño de las tarjetas de desarrollo al implementarles algoritmos metaheurísticos que resuelvan el problema propuesto.

Para la aplicación de dicha metodología, se necesita información obtenida por las tarjetas de desarrollo a travéz de índices de calidad, tales como tiempo, memoria utilizada, etc, que permita describir y analizar el comportamiento de estas al resolver los algoritmos con el fin de caracterizar las tarjetas de desarrollo.

En la figura 4.1 se muestra el diagrama de la metodología que se propone para la solucion del problema, la cual consta de cuatro etapas; entrada de datos donde se intriducen a través de instancias, aplicación de los algoritmos en las tarjetas de desarrollo, caracterización de desempeño y análisis de resultados.

En la figura 4.2 se muestra un segundo diagrama de la metodología en la cual para la etapa de entrada de datos, además de utilizar las intancias del problemas, se tienen los datos de los sensores que se van adquiriendo, además de la sotras etapas de aplicación de los algoritmos en las atrjetas de desarrollo, caracterización de desempeño y análisis de resultados; las cuales se describirán a continuación:

4.1. Entrada de datos

Para esta etapa se describe la manera en cómo se obtendrá la información la cual será dada al algoritmo para resolver el problema dentro de las tarjetas de desarrollo. Como primer paso se encuentran las instancias del problema que pueden ser tomadas de la literatura. En la figura 4.3 se observa la estructura de una instancia del problema, empezando con el nombre de esta, el tipo de problema para el cual esta diseñada, comentarios sobre donde ha sido recopilada dicha información, tamaño de la instancia, tipo de distancia que se debe utilizar para sacar las distancias entre nodos, los valores de cada nodo conformado por un identificador, un valor en x y un valor en y, por

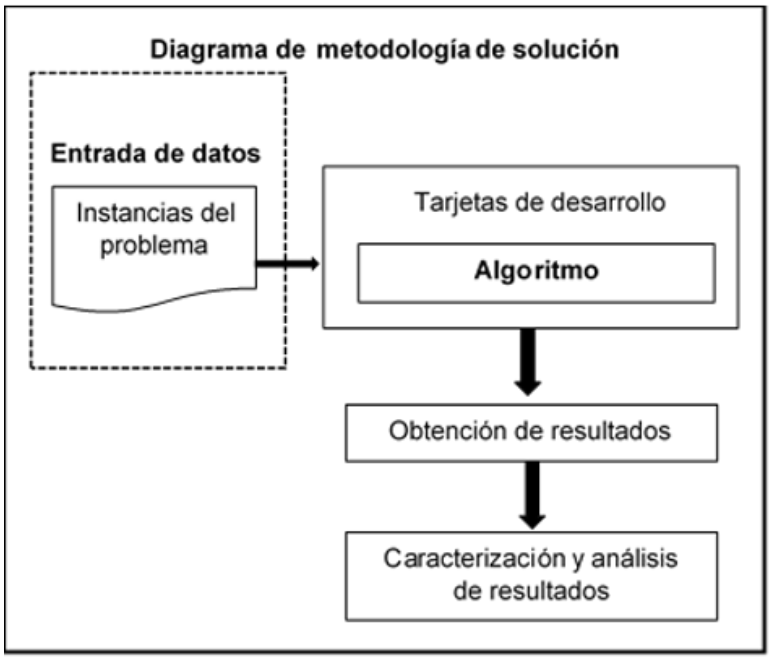

Figura 4.1: Metodología de solución

ultimo el final del documento. Dichas instancias de prueba fueron tomadas del sitio de nternet reconocido para el problema propuesto.

También se utiliza un conjunto de datos obtenidos por sensores, en la siguiente sección se muestran, que son conectados a las tarjetas de desarrollo, los cuales se usarán para medir el ambiente de desarrollo que servirá para poder observar el comportamiento de las tarjetas cuando se les agrega más componentes con los cuales deben interactuar. En la figura 4.4 se da un ejemplo de las lecturas de los sensores cuando están leyendo datos. En primera parte se da el valor de la temperatura y el de humedad, después se da el valor registrado por la flama, para obtenerlo se utiliza un factor que genere fuego, después el valor de ultrasonido, que sirve para verificar por donde va moviéndose el vehículo, por último, da un contador de movimiento, este se genera cada vez que los motores se utilizan.

4.2. Tarjetas de desarrollo

En esta etapa se conforma por como estan implementadas las tarjetas de desarrollo con los sensores y actuadores que se conectan con el entorno. En la figura 4.5 se muestra el diagrama general de la conexión entre los componentes principales; tarjetas de desarrollo, sensores y actuadores; como se puede ver los sensores son conectados para obtener información adicional que sera procesada por el algotirmo integrado, en este caso, cuando las tarjetas son puestas en movimiento, los sensores son activados de igual manera, asi cuando el algoritmo requiera de la información, los sensores ya estaran en funcionamiento y proveerán la información necesaria.

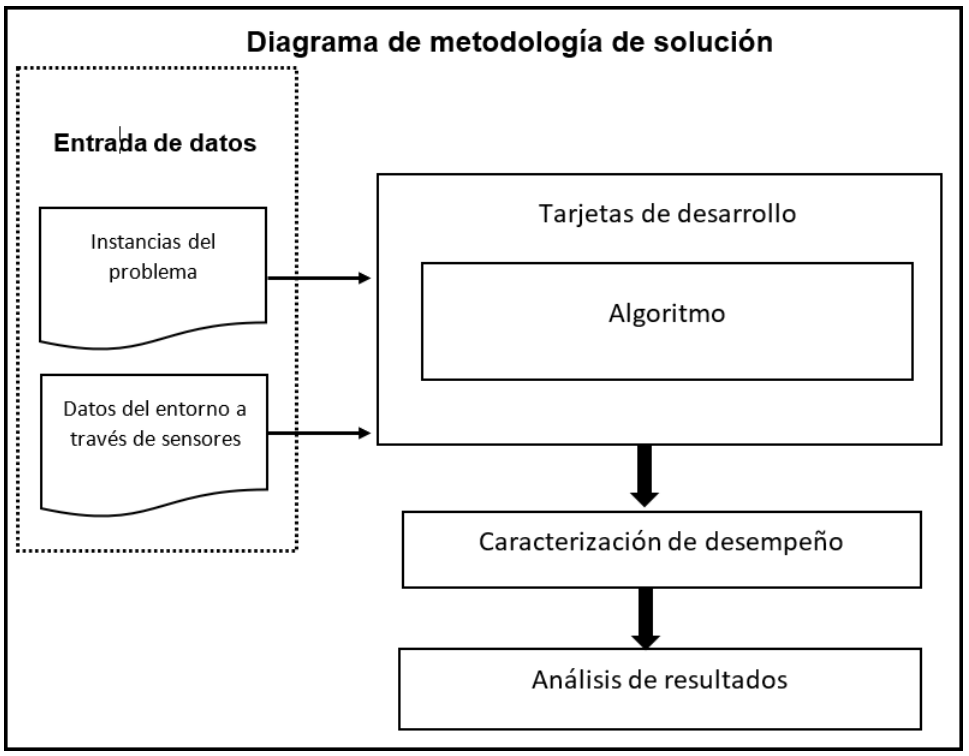

Figura 4.2: Metodología de solución

De igual forma, cuando la tarjeta es puesta en movimiento, los actuadores son encendidos en espera de recibir alguna indicacion por parte del algoritmo, para llevar a cabo acciones concretas en el entorno en respuesta a la información que los sensores mandan.

Cabe mencionar que para las diferentes tarjetas de desarrollo fue necesario hacer modificaciones y adaptaciones para poder hacer uso de cada uno de los componentes con los que se deseaba utilizar. Por lo cual acontinuación se detallan la forma en que se desarrollaron en ambas tarjetas.

Arduino

Empezando con la tarjeta arduino, en la figura 4.6 se muestra el esquema general de la estructura de los componentes, se puede observar que la mayoria de los componentes son conectados a las entradas digitales, en cuanto al controlador de motores se conecta a los pines analogos, unido a este se encunetran cuatro motores mororreductores, los cuales permiten el movimiento del dispositivo móvil.

Como se puede observar, debido a la reducida cantidad de memoria de arduino, no se puede generar archivos para guardar los resultados obtenidos, por tanto, se incorporó un modulo de Bluetooth HC-05 para poder hacer una comunicacion con una laptop en donde poder observar los datos.

Para poder adquirir los datos primero se configuró el Bluetooth HC-05, luego se

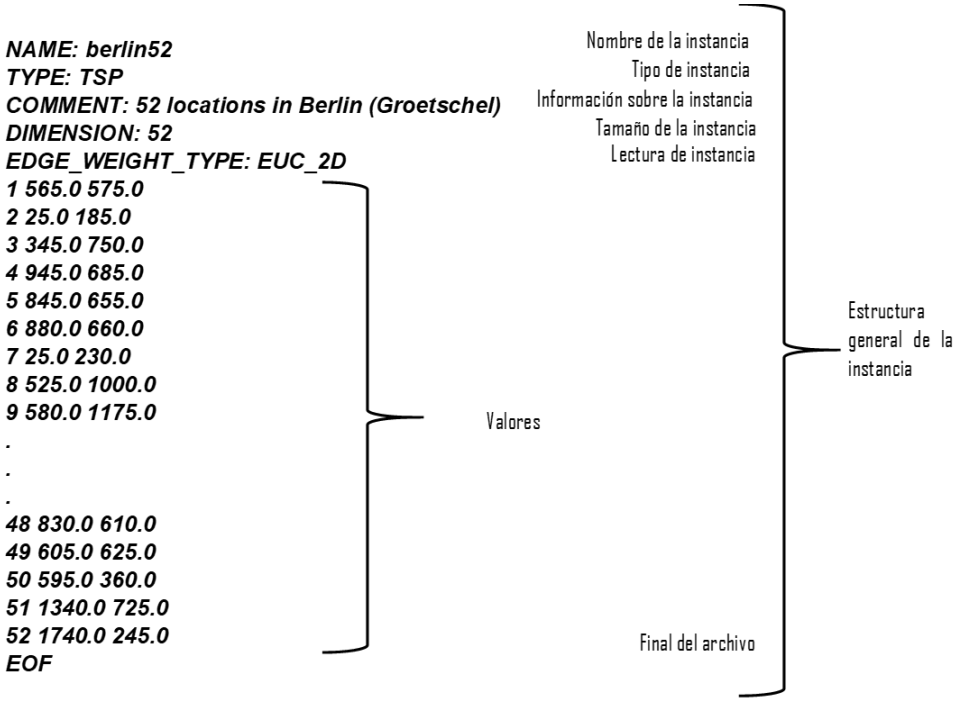

Figura 4.3: Ejemplo de instancia

Valor Temperatura: 33; Valor Humedad: 12; Valor Flama: 300; Valor Ultrasonido: 120; Movimiento: 1
Valor Temperatura: 32; Valor Humedad: 12; Valor Flama: 298; Valor Ultrasonido: 12; Movimiento: 1
Valor Temperatura: 33; Valor Humedad: 13; Valor Flama: 299; Valor Ultrasonido: 10; Movimiento: 1
Valor Temperatura: 33; Valor Humedad: 12; Valor Flama: 300; Valor Ultrasonido: 25; Movimiento: 1
Valor Temperatura: 33; Valor Humedad: 11; Valor Flama: 301; Valor Ultrasonido: 120; Movimiento: 1

Figura 4.4: Ejemplo de resultados de sensores

implementó un programa en el IDE de arduino para realizar la verificación de los datos y finalmente un programa para poder guardar estos datos y poder verlos en un archivo en Excel.

En cuanto a los algoritmos empleados en arduino y raspberry pi, se tiene estructuras diferentes en cuanto al almacenamiento de los datos, en la Tabla 4.1 se muestran las estructuras utlizadas en ambas. Esto debido a la poca memoria en Arduino es más factible utilizar vectores o variables de poco uso de memoria, así se puede obtene mejor eficiencia en cuanto a la rapidez de resultados.

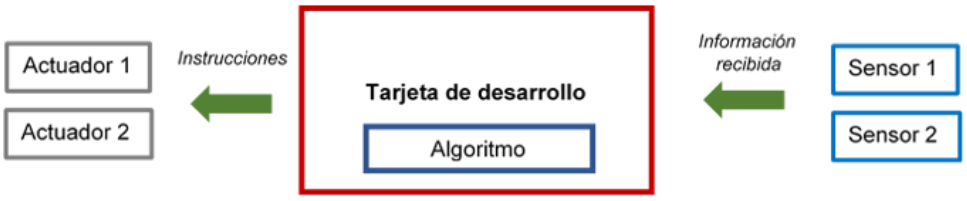

Figura 4.5: Modulo de conexión

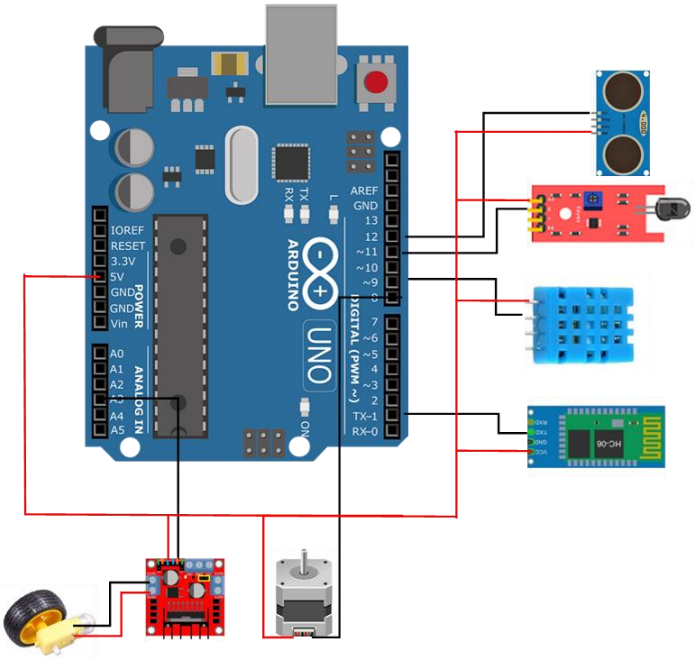

Figura 4.6: Modulo de conexión arduino

Tabla 4.1: Estructuras utilizadas en las tarjetas de desarrollo

Arduino	Descripción	Raspberry Pi	Descripción
Poblacion_i	Estructura donde se guarda las poblaciones. Vector	Poblacion_i	Estructura donde se guarda las poblaciones. Matriz
Seleccion_i	Estructura donde se guarda los candidatos a selección. Vector	Seleccion_i	Estructura donde se guarda los candidatos a selección. Matriz
Cruza_i	Estructura donde se guardan los nuevos individuos después de la cruza. Vector	Cruza_i	Estructura donde se guardan los nuevos individuos después de la cruza. Matriz
Mutación_i	Estructura donde se guardan los individuos que se le as plica la mutación. Vector	Mutación_i	Estructura donde se guardan los individuos que se le as plica la mutación. Matriz
Hormigas_i	Estructura donde se guardan las hormigas generadas. Vector	Hormigas_i	Estructura donde se guardan las hormigas generadas. Matriz
Memoria_ hormigas	Estructura donde se guardan las memorias de recorridos de las hormigas. Vector	Memoria_ hormigas	Estructura donde se guardan las memorias de recorridos de las hormigas. Matriz
Probabilidad_ por_movimiento	Estructura donde se guardan la probabilidad de las hormigas. Vector	Probabilidad_ por_movimiento	Estructura donde se guardan la probabilidad de las hormigas. Matriz
Hormigas_ actualización	Estructura donde se actualizan las hormigas en cada generación. Dos Vectores.	Hormigas_ actualización	Estructura donde se actualizan las hormigas en cada generación. Matriz

El algoritmo utilizado en la plataforma arduino se muestra a continuación, el cual empieza con la inicializacion de los sensores,en seguida se hace la validación con el sensor bluethoot hacia la computadora, esto require de un momento para hacer los ajustes necesario para poder recibir los datos de los sensores, despues se genera la población inicial, y se calcula el fitness de esta.

Una vez obtenida la población inicial, se inicia el proceso principal del algoritmo genetico, primero se pregunta cuantos nodos van hacer uso de los sensores, despues se empieza con la selección, la cruza y mutación, se calculan los fitnes de los nuevos elementos y se agregan los mejores evaluados, despues se vuelve a epezar con el proceso.

Algorithm 8: Algoritmo genético

Entrada: número de generaciones g, población p, porcentaje de cruza c, porcentaje de mutación m,
Salida : best solution $Xbest$

```
1  iniciar_sensores()
2  concatr_transmisión_datos()
3  Generar_poblacion_inicial()
4  Calcular_fitness()
5  while fin_paro do
6      foreach i ∈ p do
7          Utilizar_sensores()
8          Seleccion(i)
9          Cruza(i, c)
10         Mutacion(i, m)
11         Utilizar_sensores()
12         Calcular_fitness()
13         Agregar_mejores_individuos()
14     end
15 end
16 return xBest= seleccionarMejor(p)
```

Para el algoritmo Colonia de hormigas, se muestra a continuación su pseudocodigo. primero se inicializan la inicializacion de los sensores,en seguida se hace la validación con el sensor bluethoot hacia la computasdora, esto require de un momento para hacer los ajustes necesario para poder recibir los datos de los sensores, despues se crean las hormigas que se utilizaran para resolver el problema.

Despues se inicia el ciclo pirncipal del algoritmo, empezando por generar los movimiento posibles de las hormigas, despues se evlauan los movimenots creados y si estos son validos se le agregan a la posiscion de la hormiga en cuestión. Una vez obtenido los movimientos posibles se manda a llamar a la funcion donde se analizara en que nodo se utilizarán los sensores, despues se agregan el deposito de feromonas. Por ultimo se actualizan las soluciones y se reinicia el proceso.

Algorithm 9: Algoritmo colonia de hormigas

Entrada: número de hormigas *N*; trazo inicial de feromonas *F*; estructura de vecinada de hormigas *C*; estructura de memoria colectiva de hormigas *M*; estructura de probabilidad para cada movimiento de una hormiga *P*

Salida : best solution *Xbest*

```
1  while termination condition not met do
2      iniciar_sensores()
3      concatr_transmisión_datos()
4      Utilizar_sensores()
5      ConstructSolutionsbyAnts()
6      foreach i= 1 to N do
7          M = ActualizarMemoria(M)
8          while not Solucionconstruida do
9              Utilizar_sensores()
10             C = MovimientosPosibles(pos)
11             P = EvaluarMovimientos(C,M)
12             pos= MovimientoHormiga(P)
13             Utilizar_sensores()
14             F = DepositarFeromonas(pos)
15             x := AnnadirComponente(pos)
16         end
17         x = ActualizarSoluciones(x)
18     end
19     F = EvaporacionFeromonas(x,F)
20     F = AccionesDemonio(x,F)
21 end
22 return xBest= seleccionarMejor(x)
```

Raspberry Pi

Para la tarjeta raspberry pi, en la figura 4.7 se muestra el esquema general de la estructura de los componentes, se puede observar que la mayoria de los componentes son conectados en los pines GPIO a las entradas digitales, en cuanto al controlador de motores se instalaron las librerias relacionadas de la literatura, esto debido que tiene u funcionamientop diferente para Raspberry P, permitiendo el movimiento del dispositivo móvil.

A diferencias con ardunio, en primer lugar no es necesario utilizar el modulo Bluetooth HC-05, esto debido a que esta tarjeta cuanta con espacio suficiente en memoria para generar archivos que permiten almacenar los datos obtenidos por el algoritmo, ademas que tiene incluido un modulo de Bluethoot integrado en el modulo Wi-FI. Por lo cual para revisar los datos obtenidos solo fue necesario conctar un monitos para poder visualizarlos.

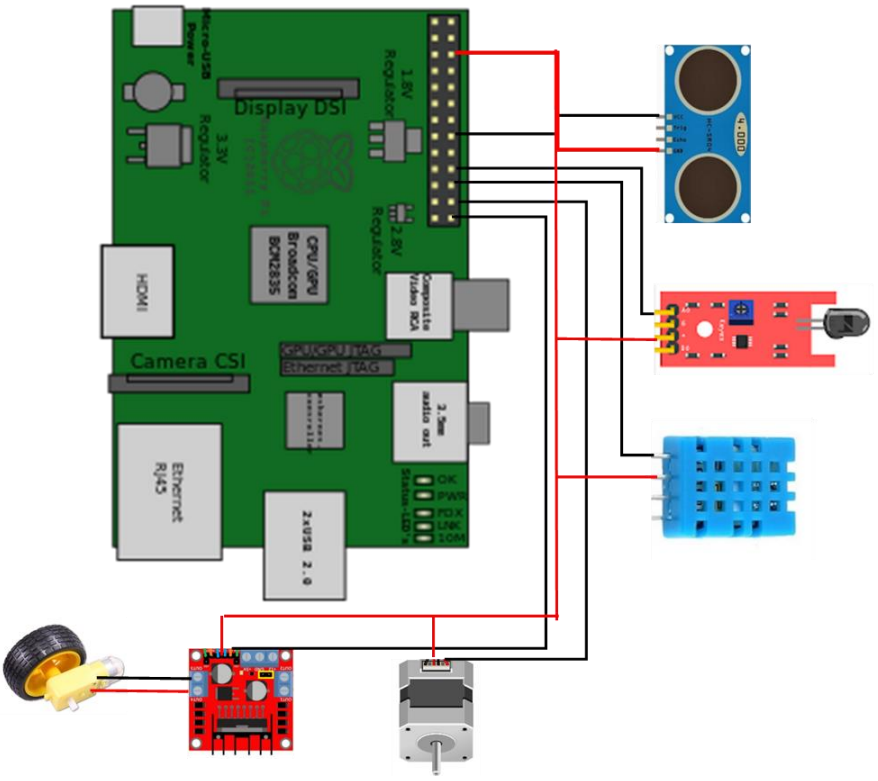

Figura 4.7: Modulo de conexión raspberry pi

En cuanto a los algoritmos empleados en raspberry pi, se muestra a continuación, el cual empieza con la inicializacion de los sensores,a diferencia de los algoritmo empleados en arduino, aqui no es necesario validar la concección con la computador, por lo cual ese paso se omite, despues se genera la población inicial, y se calcula el fitness de esta.

Una vez obtenida la población inicial, se inicia el proceso principal del algoritmo genetico, primero se pregunta cuantos nodos van hacer uso de los sensores, despues se empieza con la selección, la cruza y mutación, se calculan los fitnes de los nuevos elementos y se agregan los mejores evaluados, despues se vuelve a epezar con el proceso.

Algorithm 10: Algoritmo genetico

Entrada: número de generaciones g, población p, porcentaje de cruza c, porcentaje de mutación m,
Salida : best solution $Xbest$

```
1  Generar_poblacion_inicial()
2  Calcular_fitness()
3  while fin_paro do
4  |  foreach i ∈ p do
5  |  |  Utilizar_sensores()
6  |  |  Seleccion(i)
7  |  |  Cruza(i, c)
8  |  |  Mutacion(i, m)
9  |  |  Utilizar_sensores()
10 |  |  Calcular_fitness()
11 |  |  Agregar_mejores_individuos()
12 |  end
13 end
14 return xBest= seleccionarMejor(p)
```

Para el algoritmo colonia de hormigas, se muestra a continuación su pseudocodigo. primero se inicializan la inicializacion de los sensores, despues se crean las hormigas que se utilizaran para resolver el problema. Despues se inicia el ciclo pirncipal del algoritmo, empezando por generar los movimiento posibles de las hormigas, despues se evaluan los moviminetos creados y si estos son validos se le agregan a la posiscion de la hormiga en cuestión. Una vez obtenido losmovimientos posibles se manda a llamar a la funcion donde se analizara en que nodo se utilizarán los sensores, despues se agregan el deposito de feromonas. Por ultimo seactualizan las soluciones y se reinicia el proceso.

Algorithm 11: Algoritmo colonia de hormigas

Entrada: número de hormigas *N*; trazo inicial de feromonas *F*; estructura de vecinada de hormigas *C*; estructura de memoria colectiva de hormigas *M*; estructura de probabilidad para cada movimiento de una hormiga *P*

Salida : best solution *Xbest*

```
1  while termination condition not met do
2      Utilizar_sensores()
3      ConstructSolutionsbyAnts()
4      foreach i= 1 to N do
5          M = ActualizarMemoria(M)
6          while not Solucionconstruida do
7              Utilizar_sensores()
8              C = MovimientosPosibles(pos)
9              P = EvaluarMovimientos(C,M)
10             pos= MovimientoHormiga(P)
11             Utilizar_sensores()
12             F = DepositarFeromonas(pos)
13             x := AnnadirComponente(pos)
14         end
15         x = ActualizarSoluciones(x)
16     end
17     F = EvaporacionFeromonas(x,F)
18     F = AccionesDemonio(x,F)
19 end
20 return xBest= seleccionarMejor(x)
```

Capítulo 5

Experimentación y resultados

En este capitulo se explica la experimentación realizada para este trabajo en donde se probaron dos algoritmos, colonia de hormigas y algoritimo genético, los cuales fueron implementados en las tarjetas de desarrollo y a continuación, se explica la experimentación realizada.

Características del entorno de experimentación.
Las características de la máquina son las siguientes: marca DELL con un procesador Intel Core i5-9500M CPU con 3.00GHz, memoria RAM de 20 GB y Sistema Operativo Windows 10.

Las características de las tarjetas de desarrollo son:

- **Arduino:** modelo UNO con un microcontrolador ATMega328 con 16Mhz con memoria FLASH 32Kb.
- **Raspberry Pi:** modelo 3B+ con un procesador Broadcom BC M2837B0 DE 64Bits con memoria RAM de 1GB y sistema operativo Raspbian.

Configuración de los algoritmos.

Los algoritmos que fueron implementados para esta experimentación fueron algoritmo genético y algoritmo colonia de hormigas, ambos fueron desarrollados en lenguaje C. Las configuraciones de los algoritmos se dan en la Tabla 5.1.

Descripción de las instancias utilizadas:

En la tabla 5.2 se muestran las instancias utilizadas en la experimentación. Donde se indica el nombre de la instancia, una breve descripción, la cantidad de nodos y el valor óptimo. Las instancias fueron tomadas de la biblioteca TSPLIB.

Métricas utilizadas para la evaluación del desempeño de los algoritmos.
Las métricas aplicadas son de tiempo, memoria utilizada y solución óptima:

- Tiempo: muestra cuánto tardó la tarjeta en resolver el algoritmo, es obtenida por

Tabla 5.1: Configuración de parámetros

Parámetros	Algoritmo Genético	Algoritmo Colonia de hormigas
Corridas	30	30
Población	100	25
Generaciones	100	100
Porcentaje cruza	80 %	-
Porcentaje mutación	5 %	-
α	-	0.1
β	-	2.3
q_0	-	0.9

Tabla 5.2: Instancias utilizadas

Nombre	Descripción	No de nodos	Valor optimo
burma14	14-Staedte in Burma	14	3323
ulysses16	Odyssey of Ulysses	16	6859
berlin52	52 locations in Berlin	52	7542
gr202	Europe-Subproblem of 666-city TSP	202	40160
a280	drilling problem	280	2579
rat575	Rattled grid (Pulleyblank)	575	6773
gr666	666 cities around the world	666	294358
pr1002	1002-city problem	1002	259045

una función de tiempo definida en el algoritmo, se calcula por la media de 30 corridas y es estimada en segundos.

- Memoria utilizada: muestra la cantidad de memoria utilizada para cada tarjeta al resolver el algoritmo, se calcula por la media de 30 corridas y es expresada en kbytes.
- Solución óptima: muestra la media de las 30 repeticiones por cada instancia.

5.1. Experimentación 1. Comparación del comportamiento de las tarjetas de desarrollo al aumentar el numero de nodos implementando el algoritmo Dijkstra.

El propósito del experimento 1 es implementar un algoritmo determinista para obtener resultados, de manera que podamos describir el comportamiento de arduino y raspberry pi cuando el número de nodos en cada instancia aumenta. Se utilizan todas las intancias ya mencionadas y para cada instancia se ejecuta 30 veces, luego se calcula el promedio de los indicadores de calidad.

En la figura 5.1 se muestra el tiempo que utiliza el algoritmo Dijkstra al resolver cada instancias, se muestra que la tarjeta arduino tiene un incremento gradual conforme incrementa el número de nodos de las instancias. En cambio la tarjeta raspberry pi muestra un incremento drástico conforme va el aumento el número de nodos.

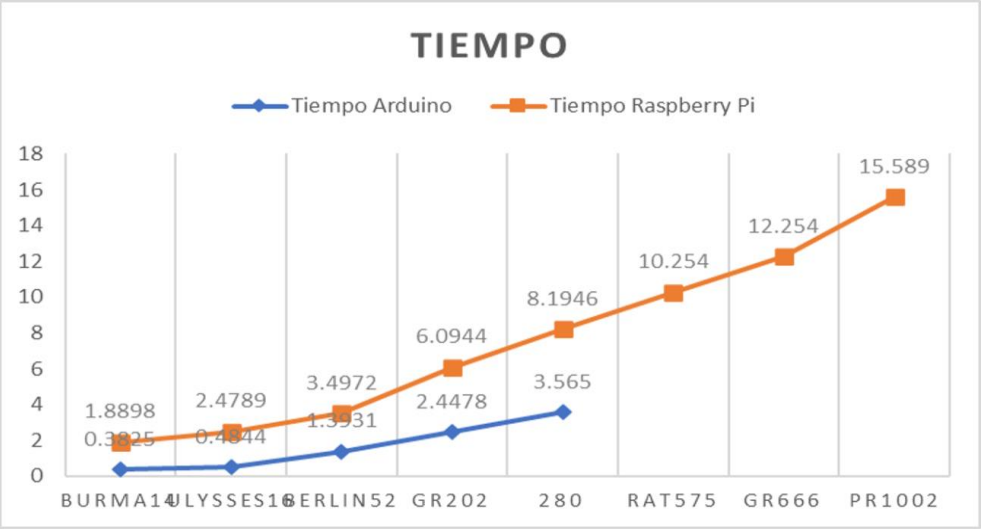

Figura 5.1: Tiempo utilizado por el algoritmo Dijkstra en ambas tarjetas.

En la figura 5.2 se muestra el uso de la memoria que utiliza el algoritmo al resolver cada instancia, se puede ver que desde la segunda instancia arduino se eleva drasticamente, esto debido al limitado recurso de la memoria. Como se pude observar a partir de la instancia 280 arduino no es capaz de resolver el algoritmo, esto debido al lmitado recurso de memoria, lo cual no da abasto para realizar más operaciones.

Con los resultados anteriores podemos concluir que en cuanto al uso de memoria la tarjeta arduino no tiene la capacidad de resolver un algoritmo determinsita con mas de 280 nodos. Por otro lado, el tiempo de ejecución en arduino es menor en comparación con el de raspberry pi, esto es debido a que arduino solo ejecuta un proceso a la vez.

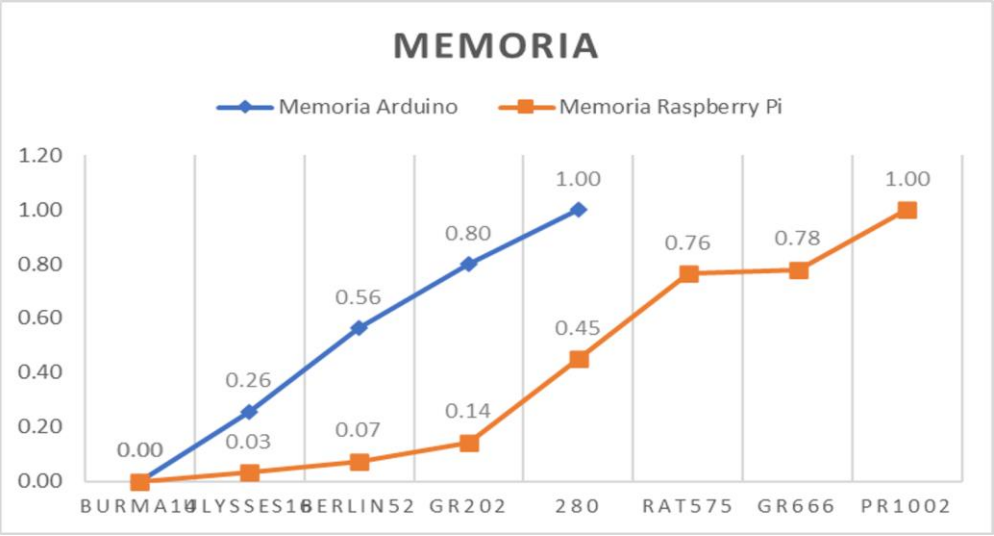

Figura 5.2: Memoria utilizada por el algoritmo Dijkstra en ambas tarjetas.

5.2. Experimentación 2. Comparación del comportamiento de las tarjetas de desarrollo al aumentar el numero de nodos implementando los algoritmos genetico y colonia de hormigas.

El propósito del experimento 2 es utilizar la configuración establecida en la Tabla 5.1 para obtener resultados, de manera que podamos describir el comportamiento de arduino y raspberry pi cuando el número de nodos en cada instancia aumenta al ejecutar el algoritmo de colonia de hormigas y el algoritmo genetico. Se utilizan todas las intancias ya mencionadas y para cada instancia se ejecuta 30 veces, luego se calcula el promedio de los indicadores de calidad.

En la figura 5.3 se muestra el tiempo que utiliza el algoritmo colonia de hormigas al resolver cada instancias, se muestra que la tarjeta arduino tiene un incremento gradual conforme incrementa el número de nodos de las instancias. En cambio la tarjeta raspberry pi muestra un incremento drástico conforme va el aumento el número de nodos.

En la figura 5.4 se muestra el uso de la memoria que utiliza al resolver cada instancia, se puede ver que en las primeras tres instancias ambas tarjetas utilizan un consumo de memoria igual, pero cuando la instancia aumenta a 202 nodos incrementa el uso de memoria por parte de la tarjeta ardunio.

Con los resultados anteriores podemos concluir que en cuanto al uso de memoria la tarjeta arduino tiene un consumo mayor a la utilizada por raspberry pi, esto debido a la poca capacidad de memoria establecida para esta. Por otro lado, el tiempo de ejecución en arduino es menor en comparación con el de raspberry pi, esto es debido a que arduino solo ejecuta un proceso a la vez.

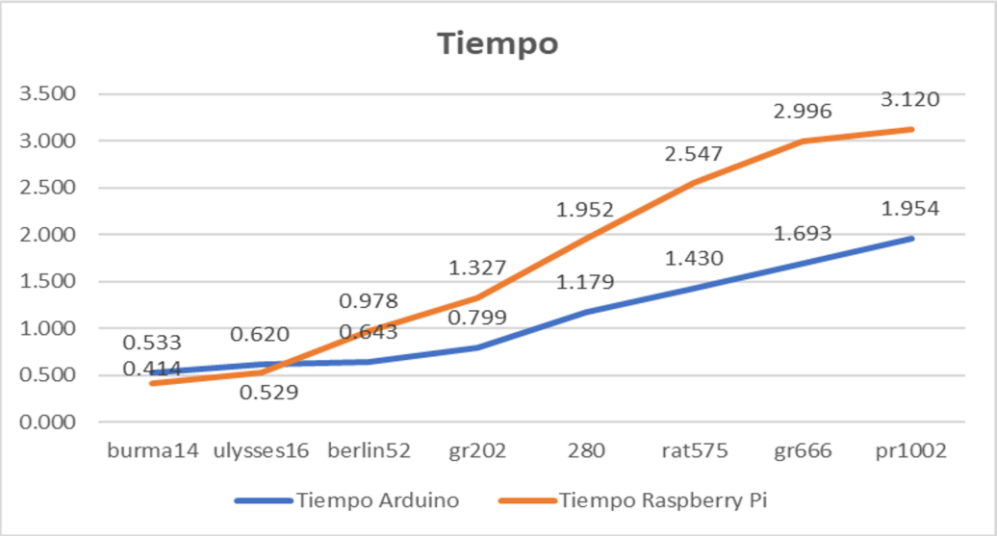

Figura 5.3: Tiempo utilizado por el algoritmo colonia de hormigas en ambas tarjetas.

Por otro lado, en la figura 5.5 se muestra el tiempo que utiliza el algoritmo genetico al resolver cada instancia, se muestra que la tarjeta raspberry pi muestra un incremento escalonado pero elevado conforme va el aumento el número de nodos. En cambio la tarjeta arduino tiene un incremento gradual conforme incrementa el número de nodos de las instancias por lo cual se mantiene bajo que el de la tarjeta raspberry pi.

En la figura 5.6 se muestra el uso de la memoria que se utiliza al resolver cada instancia, se observa que en ambas tarjetas cuando resuelven las primeras tres instancias no hay diferencia significativa, pero conforme empieza a aumentar la cantidad de nodos, la tarjeta arduino incrementa el uso de memoria, en cambio raspberry pi mantiene un uso gradual de memoria.

Con base en los resultados anteriores, podemos concluir qué en términos de uso de memoria, debido a su baja capacidad de almacenamiento, el consumo de las tarjetas arduino es superior al de las tarjetas raspberry pi. Por otro lado, el tiempo de ejecución de arduino es menor que el de raspberry pi, porque arduino solo ejecuta un proceso a la vez.

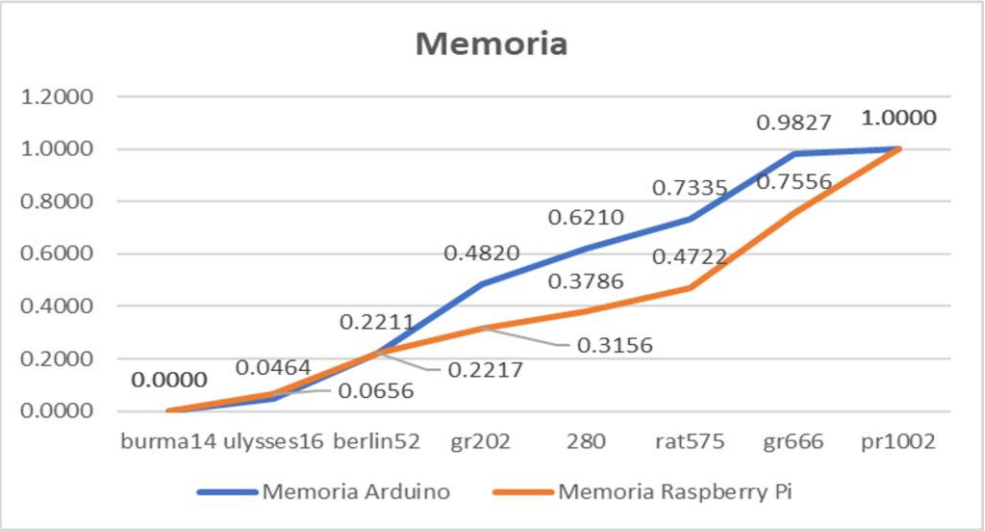

Figura 5.4: Memoria utilizada por el algoritmo colonia de hormigas en ambas tarjetas.

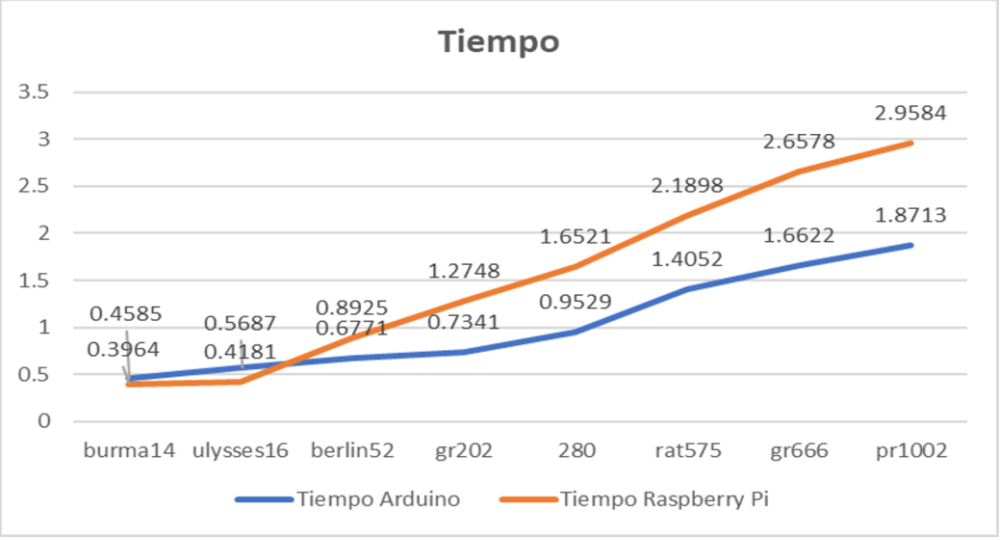

Figura 5.5: Tiempo que tardó el algoritmo genético en ambas tarjetas.

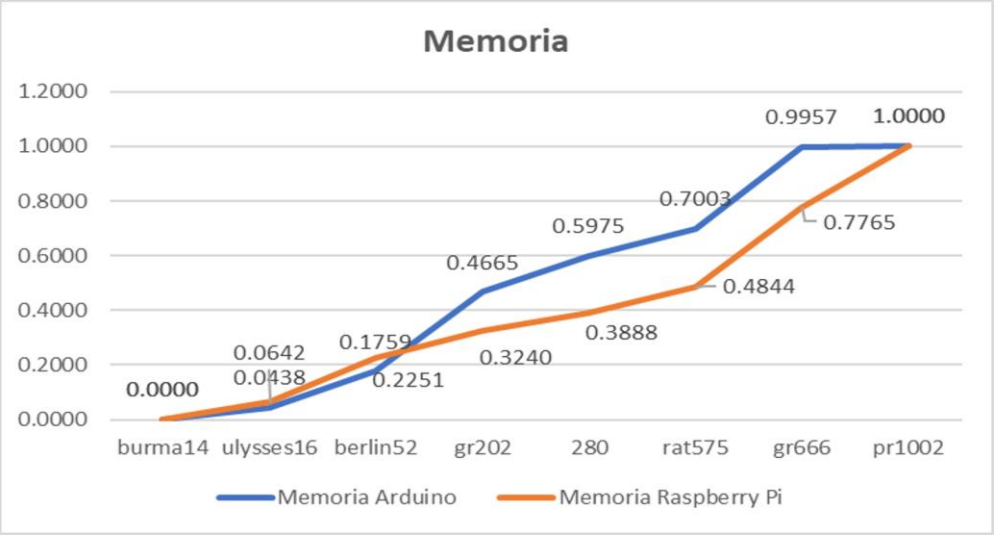

Figura 5.6: Memoria utilizada por el algoritmo genético en ambas tarjetas.

5.3. Experimentación 3. Comparación del comportamiento de las tarjetas de desarrollo utilizando sensores al implementar los algoritmos genetico y colonia de hormigas.

El propósito del experimento 3 es poder describir el comportamiento de arduino y raspberry pi al resolver los algoritmos colonia de hormigas y genetico, cuando se les implementan dos sensores que influyen en la búsqueda de las soluciones. El propósito de los sensores es darle instrucciones de ejecución a las tarjetas cuando están en ciertos nodos, el uso de los sensores es a traves de una probabilidad, por lo cual no todos los nodos pueden ser utilizados.

Además de observar a las tarjetas de desarrollo cuando aparte de tener los sensores, como trabajan cuando aumenta el número de nodos de las instancias. Así para esta parte se usan todas las instancias ya mencionadas, donde a cada instancia se ejecuta 30 veces y después calcule el promedio de los indicadores de calidad.

Para esta experimentación se implementó el algoritmo colonia de hormigas en ambas tarjetas de desarrollo, utilizando dos sensores, los cuales están implementados dentro del vehículo móvil. El objetivo de esta experimentación es observar el comportamiento de las tarjetas de desarrollo al utilizar más componentes. Se realizaron 30 corridas para cada instancia. Después, se calculó las medias de las métricas de calidad.

En la figura 5.7 se muestra el tiempo que se utiliza al resolver cada instancia, se muestra que la tarjeta raspberry pi tiene un incremento gradual conforme va el aumento el número de nodos. En cambio, la tarjeta arduino tiene un incremento muy rápido por lo cual supera al tiempo utilizado por el de la tarjeta raspberrry pi en la mayoría de las instancias.

En cuanto al uso de memoria, se puede ver en la figura 5.8 que la tarjeta arduino tiene un incremento muy rápido a partir de la tercera instancia, en cambio, la tarjeta raspberry pi tiene un incremento gradual en todas las instancias. Cabe mencionar que la tarjeta ardunio no pudo resolver la ultima instancia *"pr*1002" debido a que se consumió la memoria disponible de esta, y no mostraba algún resultado.

Con base en los resultados mostrados, podemos concluir que en términos de uso de memoria, debido a su baja capacidad de almacenamiento, el consumo de la tarjeta arduino suele ser mayor al tener que repartirla entre los componentes utilizados y tratar de resolver el algoritmo, con lo cual en algunas instancias no logra mostrar resultados debido a que es insuficiente para seguir ejecutando el proceso, en cambio raspberry pi muestra un uso de memoria alto pero por su capacidad si logra obtener resultados.

Por otro lado, el tiempo de ejecución de arduino se vuelve mayor que el de raspberry, al tratar de interactuar con los componentes instalados y resolver el algoritmo,

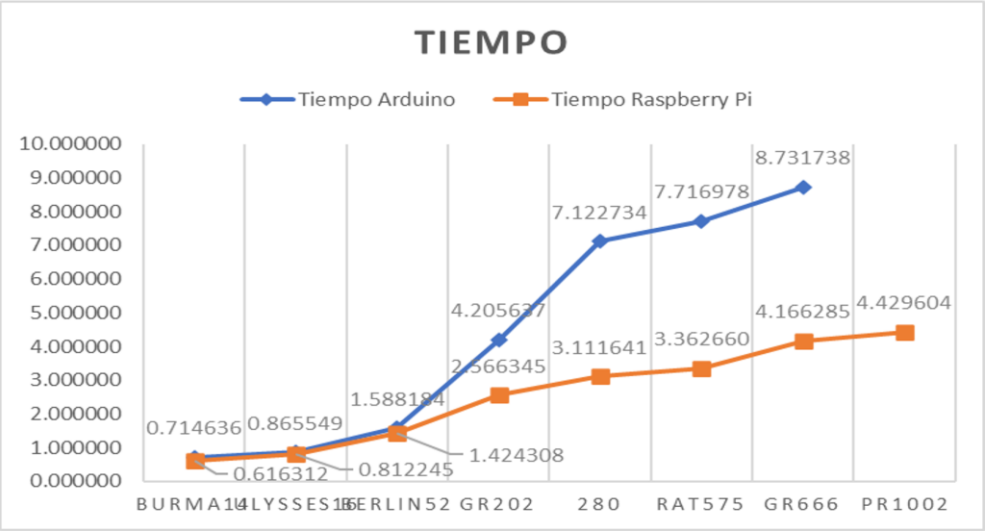

Figura 5.7: Tiempo utilizado por el algoritmo colonia de hormigas en ambas tarjetas.

en cambio para la tarjeta raspberry pi el tiempo suele crecer de manera gradual.

Tambien, en la figura 5.9 se muestra el tiempo que utiliza el algoritmo genetico al resolver cada instancia, se muestra que la tarjeta raspberry pi tiene un incremento gradual conforme va el aumento el número de nodos. En cambio, la tarjeta arduino tiene un incremento muy rápido por lo cual supera al tiempo utilizado por el de la tarjeta raspberry pi en la mayoría de las instancias.

En términos de uso de memoria, se puede ver en la Figura 5.10 que la tarjeta arduino está aumentando muy rápidamente a partir de la tercera instancia, mientras que la tarjeta raspberry pi está aumentando gradualmente en todos los casos. Vale la pena señalar que la tarjeta arduino no pudo resolver la última instancia *"pr*1002*"* porque se usó la memoria disponible y no se mostraron resultados.

Con base en los resultados mostrados, podemos concluir que en términos de uso de memoría, debido a su pequeña capacidad de memoria, el consumo de la tarjeta arduino suele ser mayor, ya que tiene que distribuirse entre los componentes utilizados e intenta resolver el algoritmo tanto que en la última instancia no se muestran resultados porque el proceso no tiene suficiente memoria para ejecutarse. Por otro lado, raspberry tiene un alto uso de memoria pero obtiene resultados debido a su capacidad.

Por otro lado, al intentar interactuar con los componentes instalados y resolver algoritmos, el tiempo de ejecución de arduino se vuelve más largo que el de raspberry, mientras que para las placas raspberry pi, el tiempo suele incrementarse gradualmente.

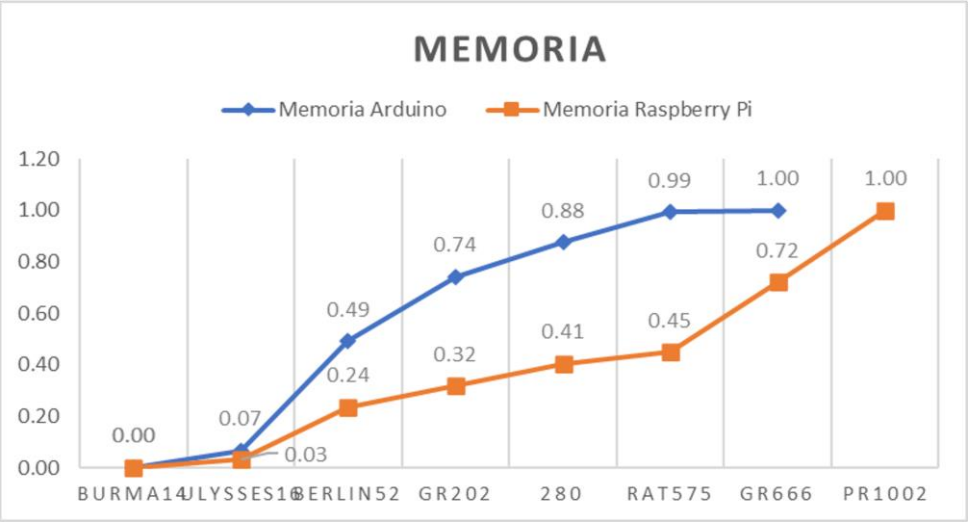

Figura 5.8: Memoria utilizada por el algoritmo colonia de hormigas en ambas tarjetas.

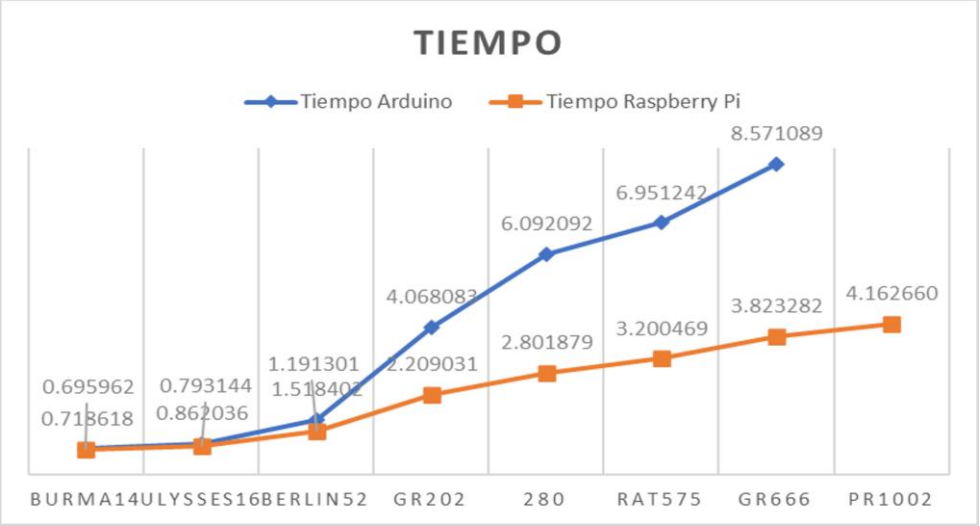

Figura 5.9: Tiempo que tardó el algoritmo genético en la tarjeta arduino.

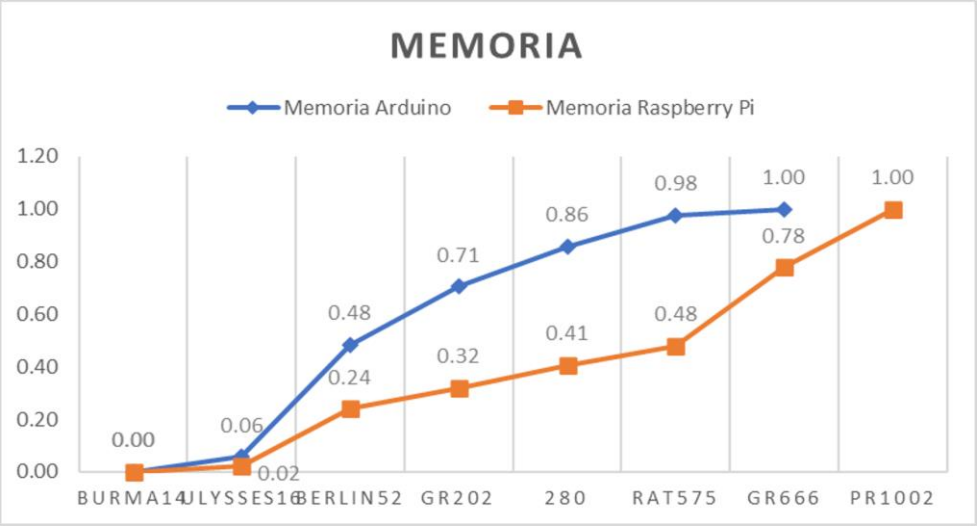

Figura 5.10: Memoria utilizada por el algoritmo genético en la tarjeta arduino.

Comentarios finales

En la experimentación con un algoritmo determinista se puede observar que arduino es capaz de resolver el problema con un tiempo más eficaz que el de raspberry pi, esto debido a que su unica funcion es correr un archivo, a diferencia de raspberry pi que procesa tareas simultaneamente.

Tambien se puede decir que en el uso de memortia raspberry pi es superior a arduino debido a la cantidad de memoria extra para uso, ademas se pudo demostrar el limite maximo de nodos que arduino puede resolver, siendo 280 nodos para el problema TSP-RA.

De la segunda experimentación se observa que el comportamiendo del algoritmo colonia de hormigas supera en tiempo al algoritmo genético en ambas tarjetas, sin embargo, cuando se habla en términos de memoria, el algoritmo genético utiliza menor cantidad en la tarjeta ardunio en 75 % de las instancias a diferencia del algoritmo colonia de hormigas.

En cambio, cuando pasa a la tarjeta raspberry pi el algoritmo colonia de hormigas utiliza menor memoria en 65 % de las instancias en comparacion del algoritmo genético. Se identificó que la tarjeta arduino con instancias menores a 50 nodos suele obtener un mejor desempeño en cuestión de tiempo y memoria mejor en comparación con la tarjeta raspberry pi.

También se observó que al agregarles los sensores a las tarjetas de desarrollo el comportamiento del algoritmo genético es menor que el algoritmo colonia de hormigas en ambas tarjetas de manera similar cuando se realizó la experimentación sin sensores.

Para la tarjeta arduino en ambos algoritmos no logra resolver la última instancia por falta de memoria disponible, pero el algoritmo genético refleja que utiliza menor cantidad de memoria en comparación con el algoritmo colonia de hormigas.

Para la tarjeta raspberry pi se observa que el consumo de memoria por parte del algoritmo colonia de hormigas es menor en dos instancias, sin embargo en tres instancias ambos algoritmos consumen la misma cantidad. Se identifico que en la tarjeta arduino con instancias mayores a 50 nodos suele obtener un desempeño, en cuestión de tiempo y memoria, deficiente en comparación con la tarjeta raspberry pi.

Capítulo 6

Conclusiones y trabajo futuro

Este capítulo presenta las conclusiones y los resultados finales como resultados de la investigación y propone sugerencias para trabajos futuros.

6.1. Conclusiones

En este trabajo de tesis, se cumplio satisfactoriamente con los objetivos planeados mediante una metodología de análisis experimental de algoritmos para obtener explicaciones sobre el desempeño observado por cada tarjeta al resolver un algoritmo que da solucion al problema de interés.

La metodología propuesta fue evaluada al aplicarse en el análisis de los resultados obtenidos por las tarjetas de desarrollo, en donde el conocimiento obtenido permitió identificar los limites, comportamiento, los rangos de soluciones y tiempo de respuesta de las tarjetas de desarrollo evaluadas.

Entre las principales aportaciones de este trabajo obtenidos mediante los resultados experimentales se encuentran:

1. Se dealizo un analisis de los modelos de las arquitecturas arduino y raspberry pi para hacer una comparacion de caracteristicas particulares.
2. Se desarrolló una metodología para el análisis del desempeño de las tarjetas de desarrollo que permite comprender el comportamiento.
3. Se realizó un estudio sobre la estructura del problema TSP, el cual fue adaptado a un problema propuesto bajo ciertas caracteristicas para su solución.
4. Se definió un conjunto de indicadores de caracterización para las tarjetas de desarrollo.
5. Se desarrolló dos algoritmos metaheuristicos para la solución del problema que fuero probados en dos tarjetas de desarrollo de diferentes capacidades.

6.2. Trabajo futuro

Para dar continuidad al trabajo de investigación presentado se proponen los siguientes trabajos:

1. Desarrollar algoritmos metaheristicos diferentes a los propuestos que den solución al problema propuesto.
2. Evaluar diferentes instancias de la literatura que se pueden incorporar al problema.
3. Complementar la metodología perimitiendo que se pueda adaptar a otras tarjetas de desarrollo e incorporación de más elementos instalados en ellas.
4. Aplicar la metodología a otra clase de problemas de ruteo.

Referencias

Akbar, M. A., y cols. (2014). Simulation of fuzzy logic control for dc servo motor using arduino based on matlab/simulink. En *2014 international conference on intelligent autonomous agents, networks and systems* (pp. 42–46).

Aracelis, A., Casco, M. R., N., y Pinzón, C. (2020). Prototipo de sistema experto basado en lógica difusa para la monitorización del ruido en espacios educativos.

Arduino. (s.f.). *Arduino uno rev3.* https://store.arduino.cc/usa/arduino-uno-rev3. ([Web; accedido el 06-08-2019])

Arvizu, V., Carlos, Liñan, R., Angel, y Torres Treviño, L. M. (2016). Controlador difusivo-evolutivo para robot seguidor de luz. *Ingenierías*, *19*(72), 35–52.

Azi, N., Gendreau, M., y Potvin, J.-Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, *202*(3), 756-753.

Barcos, L., Rodríguez, V., Álvarez, M. J., y Robusté, F. (2002). Algoritmo basado en la optimización mediante colonias de hormigas para la resolución del problema del transporte de carga desde varios orígenes a varios destinos. En *V congreso de ingeniería del transporte angel ibeas portilla-josé mP díaz y pérez de la lastra© santander-cit.*

Barr S., K. P. R. M. S. R., Golden L. (1995). Designing and reporting on computational experiments with heurístic methods. *Journal of Heuristics*, *1*(1), 2-32.

Bibiana, L., Medina, R., Cristina, E., Rota, G. L., Arturo, J., y Castro, O. (2011). Una revisión al estado del arte del problema de ruteo de vehículos: Evolución histórica y métodos de solución. *Universidad Distrital Francisco José de Caldas*, *16* (2), 35-55.

Cabezas, I. (2017). Aplicación de path planning de un robot móvil basado en raspberry pi.

Coello, C. (2016). Introducción a la computación evolutiva. *Instituto Politécnico Nacional*.

Coronel, J. M. P., Santacruz, H. D. M., Ríos, C. G. L., y Cegla, B. B. (2006). Algoritmos de optimización multi-objetivos basados en colonias de hormigas. En *Latin-american conference on informatics (clei)* (Vol. 10).

Dantzig, F. D. J. S., G. (1954). Solution of a large scale traveling salesman problem. *Operations Research 2*.

Deng, Y., Chen, Y., Zhang, Y., y Mahadevan, S. (2012). Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, *12*(3), 1231–1237.

Eiben, A. E., Smith, J. E., y cols. (2003). *Introduction to evolutionary computing*

(Vol. 53). Springer.
Ferdoush, S., y Li, X. (2014). Wireless sensor network system design using raspberry pi and arduino for environmental monitoring applications. *Procedia Computer Science*, *34*, 103–110.
Garcia, A. (2015). *MOTOR DE CORRIENTE CONTINUA - Alvaro garcia naranjo.* Descargado 2020-05-30, de https://sites.google.com/site/alvarogarcianaranjo/mootor-de-corriente-continua
Gómez, F. (2017). Vehículo autónomo para gestión de trayectorias basado en arduino y raspberry.
Guerra, J. A. M., Valadez, J. M. C., Soberanes, H. J. P., y de Montserrat Ortiz-Aguilar, L. (s.f.). Un acercamiento a los algoritmos as y aco resolviendo el tsp.
Herrador, R. E. (2009). Guía de usuario de arduino. *Universidad de Córdoba*, *13*.
Hincapié, R. A., Porras, C. A. R., y Gallego, R. A. (2004). Técnicas heurísticas aplicadas al problema del cartero viajante (tsp). *Scientia et technica*, *10* (24), 1–6.
Holland. (1975). Adaption in natural and artificial systems. *University of Michigan Press*.
Jasika, N., Alispahic, N., Elma, A., Ilvana, K., Elma, L., y Nosovic, N. (2012). Dijkstra's shortest path algorithm serial and parallel execution performance analysis. En *2012 proceedings of the 35th international convention mipro* (pp. 1811–1815).
Khalaf, O. I., Abdulsahib, G. M., y Zghair, N. A. K. (2019). Iot fire detection system using sensor with arduino. *AUS*, *26*, 74–78.
Latha, N. A., Murthy, B. R., y Kumar, K. B. (2016). Distance sensing with ultrasonic sensor and arduino. *International Journal of Advance Research, Ideas and Innovations in Technology*, *2*(5), 1–5.
Núñez, F. P. R., y Dietrich, N. B. (s.f.). Aplicación de una nueva variante de algoritmo basado en ant colony optimization (aco) al tsp many-objective.
Olivera, A. (2004). Heurísticas para problemas de ruteo de vehículos. *Universidad de la República*.
Pawar, P. A. (2014). Heart rate monitoring system using ir base sensor & arduino uno. En *2014 conference on it in business, industry and government (csibig)* (pp. 1–3).
Pi, R. (2015). Raspberry pi 3 model b. *online].(https://www. raspberrypi. org* .
Quiroz, M. (2009). Caracterización de factores de desempeño de algoritmos de solución de bpp. *Informe]: Tesis de Maestría.-Instituto Tecnológico de Ciudad Madero:[sn]*.
Raspberry. (s.f.). *Raspberry documentation.* https://www.raspberrypi.org/documentation/. ([Web; accedido el 06-08-2019])
Rodríguez, M. J. (2018). Raspberry pi como plataforma de algoritmos de machine learning: reconocimiento de imágenes y datos financieros en streaming.
Salas, A. (2008). Acerca del algoritmo de dijkstra. *arXiv preprint arXiv:0810.0075* .
Toth, P., y Vigo, D. (2002). The vehicle routing problem. *Society of Industrial and Applied Mathematics (SIAM) monographs on discrete mathematics and applications*, *109*, 1-23.
Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly 21*.

Anexo A

A continuación se describen los sensores implementados en Raspberrry Pi. Como primer paso se muestra el codigo del senso de flama, empezando por el llamado al uso de GPIO de la tarjeta, Despues de declara el pin donde se conecta el sensor, con el metodo callback inicializamos el uso del sensor. despues con los metodos detected y callback leemos los datos detectados por el sensor y los mostramos en pantalla. En la figura 6.1 se muestra el codigo.

```python
#!/usr/bin/python
import RPi.GPIO as GPIO
import time

#GPIO SETUP
channel = 21
GPIO.setmode(GPIO.BCM)
GPIO.setup(channel, GPIO.IN)

def callback(channel):
    print("flame detected")

GPIO.add_event_detect(channel, GPIO.BOTH, bouncetime=300)
GPIO.add_event_callback(channel, callback)

# infinite loop
while True:
        time.sleep(1)
```

Figura 6.1: Código para el sensor de flama.

En la figura 6.2 se muestra el codigo implementado para el uso del sensor de flama. primero se define el pin de la GPIO que se utiliza, despues se hace uso del metodo trigger, que es el encargado de hacer funcionar al sensor de humedad. despues declaramos la variable del tipo de sensor, la cuan contiene los metodos que brinda el sensor, asi en las variables humedity y temperature se llaman a loa metodos respectivos del sensor, finalmente se imprimen los valores que recogen del sensor.

En la figura 6.3 se muestra el codigo implementado para el uso del sensor de ultrasonido. primeramente se definen los pines donde se escucha y se distapara el ultrasonido, despues se guardan en la clase GPIO. Una vez definidas las variables podemos hacer uso de los métodos de la clase del sensor, por lo cual al utilizar el metodo output, podemos mandarle los parametros, donde le indicamos si queremos que dispare el ultasonido o si queremos que escuche lo que esta recibiendo. Asi, primero hacemos que dispare el ultrasonido y despues que escuche lo que esta leyendo„ una ves recibido el dato, se pasa por una formula que da como resultado la distancia quee sta tomando el senor, finalmente lo imprimimos en pantalla.

```
pi = pigpio.pi()
# Setup the sensor
dht22 = DHT22.sensor(pi, 4) # use the actual GPIO pin name
dht22.trigger()


# We want our sleep time to be above 2 seconds.
sleepTime = 3
def readDHT22():

# Get a new reading
dht22.trigger()

# Save our values
humidity = '%.2f' % (dht22.humidity())
temp = '%.2f' % (dht22.temperature())
return (humidity, temp)
while True:
humidity, temperature = readDHT22()
print("Humedad : " + humidity + "%")
print("Temperatura : " + temperature + "C")
sleep(sleepTime)
```

Figura 6.2: Código para el sensor de humedad y temperatura.

```
TRIG = 23
ECHO = 24
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

try

    while True:

        GPIO.output(TRIG, GPIO.LOW)
        time.sleep(0.5)

        GPIO.output(TRIG, GPIO.HIGH)
        time.sleep(0.00001)
        GPIO.output(TRIG, GPIO.LOW)

        while True:
            pulso_inicio = time.time()
            if GPIO.input(ECHO) == GPIO.HIGH:
                break

        while True:
            pulso_fin = time.time()
            if GPIO.input(ECHO) == GPIO.LOW:
                break

        duracion = pulso_fin - pulso_inicio

        distancia = (34300 * duracion) / 2

        print( "Distancia: %.2f cm" % distancia)

finally:
    # Reiniciamos todos los canales de GPIO.
    GPIO.cleanup()
```

Figura 6.3: Código para el sensor de ultrasonido.