Algoritmo de optimización colonia de hormigas para la generación automática de horarios.

Colony of optimization algorithm for the automatic generation of schedules.

Oswaldo de Jesús Chacón Pérez (1).
Estudiante, Instituto Tecnológico de Tuxtla Gutiérrez, Tecnológico Nacional de México.
oswaldochacon1606@gmail.com.

Jennifer Vanessa Aguilar Padilla (2). Estudiante, I. T. de Tuxtla Gutiérrez, Tecnológico Nacional de México.
jennifer.aguilar.padilla@gmail.com.

Octavio Ariosto Ríos Tercero* (3). I. T. de Tuxtla Gutiérrez, Tecnológico Nacional de México.
oriost@ittg.edu.mx.

Rosy Ilda Basave Torres (4). I. T. de Tuxtla Gutiérrez, Tecnológico Nacional de México.
rbasave@ittg.edu.mx.

Roberto Cruz Gordillo (5). I. T. de Tuxtla Gutiérrez, Tecnológico Nacional de México.
rcruz@ittg.edu.mx.

*corresponding author.

Artículo recibido en noviembre 12, 2019; aceptado en diciembre 10, 2019.

Resumen.

El presente trabajo aborda un problema de horarios en donde se pretende programar una serie de proyectos para un foro de propuestas de proyectos de investigación para poder ser expuestos y evaluados por un jurado en un periodo de tiempo disponible. El método utilizado fue una variante del algoritmo de optimización colonia de hormigas (ACO por sus siglas en inglés) llamado Max-Min Ant System. Se realizó un análisis del problema para identificar las variables y los factores que intervienen en el proceso. El algoritmo ACO cuenta con una fórmula, por lo que fue necesario entender y darle un significado a cada uno de los elementos que lo conforman y asociarlos con las variables del problema planteado. Los resultados obtenidos indican que es posible solucionar el problema de asignación de horarios mediante el algoritmo ACO logrando así una solución factible, en un tiempo de procesamiento bastante corto.

Palabras clave: Optimización colonia de hormigas, *Max-Min Ant System*, solución factible.

Abstract.

This paper addresses a problem of schedules which aims to schedule a series of projects for a forum of research projects proposals, this in order to be exposed and evaluated by a jury in a period of available time. The method used was a variant of the Ant Colony Optimization algorithm (ACO) called Max-Min Ants System. An analysis of the problem was carried out to identify the variables and factors involved in the process. The ACO algorithm has a formula, so it was necessary to understand and give meaning to each of the elements that make it up and associate

them with the variables of the posed problem. The obtained results indicate that it is possible to solve the problem of assigning schedules using the ACO algorithm achieving a feasible solution, in a fairly short processing time.

Keywords: Ant colony optimization, Max-Min Ant System, feasible solution.

1. Introducción.

El problema de asignación de horarios es un tipo de problema de programación que todo centro educativo presenta periódicamente debido a que en ella se pretende programar las actividades que se realizarán a lo largo del periodo escolar; consiste básicamente en programar un evento (clase, examen, etc) en una cantidad limitada de espacios de tiempo en donde pueda desarrollarse dicho evento, cumpliendo con las restricciones que el problema conlleva. Las restricciones implicadas pueden ser clasificadas en restricciones duras (restricciones que no deben ser violadas por ninguna circunstancia, por ejemplo, el empalme de horas de un evento) y restricciones suaves (restricciones que se desean satisfacer pero no son obligatorias).

Dentro de la literatura podemos encontrar que este problema ha sido tema de interés desde hace ya algunos años (Schaerf, 1999), debido al grado de dificultad que el proceso representa y es que forma parte de los problemas de tipo NP-Hard (Dorigo & Stützle, Ant Colony Optimization, 2004) (NP es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista y NP-Hard son problemas que son al menos tan difíciles como los problemas más difíciles en NP).

Por ello se han realizado numerosas investigaciones y se han propuesto diferentes métodos para encontrar soluciones que satisfagan la mayor cantidad de restricciones presentadas y reducir el tiempo de ejecución del proceso. La mayoría de los métodos que se han propuesto son algoritmos metaheurísticos que incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otros. Algunos ejemplos de los algoritmos metaheurísticos más conocidos son: Algoritmos Genéticos (Peñuela, Toro, & Franco, 2008), Búsqueda Tabú (Peñuela, Toro, & Franco, 2008), Recocido Simulado y Optimización Colonia de Hormigas (Peñuela, Toro, & Franco, 2008) (ACO por sus siglas en inglés); estos algoritmos tienen como resultado una solución factible, mas no siempre una solución óptima.

En este trabajo se hará más énfasis al algoritmo de optimización colonia de hormigas, se hablará sobre en qué consiste dicho algoritmo y de qué manera trabaja, además de explicar cómo puede ser implementado en el problema de asignación de horarios para un foro de propuestas de proyectos de investigación, en donde un conjunto de proyectos deberán ser asignados a un espacio de tiempo y a un salón para poder ser presentado y evaluados por un jurado, empezando por entender la formula con la que cuenta y brindarle una interpretación en base en el problema, además de realizar las pruebas pertinentes que demuestren que el problema se ha resuelto con una solución factible.

2. Métodos.

2.1 Algoritmo de optimización Colonia de Hormigas.

El algoritmo de optimización Colonia de Hormigas, es un método metaheurístico introducido por Marco Dorigo a principios de 1990 (Dorigo & Blum, Ant colony optimization theory: A survey, 2005), el algoritmo está inspirado en el comportamiento natural de las hormigas en la búsqueda de sus alimentos. Dicha búsqueda consiste en explorar alrededor de la colonia de manera aleatoria (ya que las hormigas no cuentan con una buena visibilidad) para encontrar alguna fuente de alimento. Una vez que han encontrado la fuente de alimento regresan al nido; durante ese regreso, las hormigas van depositando por el camino una sustancia llamada feromona (la feromona es una sustancia química que algunos animales segregan de ellos mismos y que cuya liberación puede llegar influenciar el comportamiento de los demás animales de la misma especie) y dicha feromona sirve para auxiliar al resto de las hormigas a seguir el mismo camino y encontrar la misma fuente de alimento. Cabe mencionar que el camino que las hormigas seguirán es el que mayor cantidad de feromonas tiene. Sin embargo, ¿cómo un camino podría tener mayor cantidad de feromona que

otra? considerando la distancia que existe entre la fuente de alimento y la colonia, entre más corto sea el camino, mayor cantidad de feromonas habrá en él porque habrá mayor cantidad de hormigas recorriéndola en comparación al camino más largo en un mismo lapso de tiempo. Se debe considerar que entre más tiempo transcurra sin que una hormiga pase por algún camino, la evaporación hará que ese camino sea cada vez más indeseable**.** Esta característica de las hormigas es la que permite que pueda ser propuesta como una manera de solucionar problemas de optimización reduciendo el tiempo de ejecución.

2.2 Descripción del algoritmo.

El algoritmo trabaja a partir de hormigas artificiales, que serán las encargadas de generar por cada iteración una solución factible al problema que se pretende resolver. Cada una construye su solución a partir de un grafo en donde cada nodo representa las posibles opciones que la hormiga podría elegir. La conexión que existe entre dos nodos, contiene información que las hormigas deben utilizar para poder elegir una de ellas, las cuales son (Dorigo & Stützle, Ant Colony Optimization, 2004):

- Información heurística: representa la información a priori sobre la instancia del problema o información en tiempo de ejecución proporcionada por una fuente diferente de las hormigas.
- Información de rastros de feromonas artificiales: se mide la deseabilidad aprendida en el movimiento de un nodo a otro, lo cual busca imitar la feromona real que depositan las hormigas naturales. Esta información es modificada mientras que se ejecuta el algoritmo dependiendo de las soluciones encontradas por los insectos.

2.2.1 Probabilidad de elegir un nodo.

Para poder elegir un nodo es necesario calcular las probabilidades que tiene cada uno y para ello, el algoritmo cuenta con la siguiente fórmula para calcularlo:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}\,[n_{ij}]^{\beta}}{\sum l \in N_i^k\,[\tau_{il}]^{\alpha}[n_{il}]^{\beta}} \quad \text{Sí } j \in N_i^k \tag{1}$$

donde τ_{ij} = es el valor del rastro de feromonas en la arista del nodo i al nodo j, $n_{ij} = 1/d_{ij}$ es un valor heurístico que está disponible a priori, d_{ij} representa la "distancia" que existe entre dos nodos (la distancia puede tomar diferentes interpretaciones según el problema), α y β son dos parámetros (α con valor de [0,1] y $\beta > 0$) que determinan la influencia relativa del rastro de feromonas y la información heurística respectivamente, y N_i^k es el nodo factible de la hormiga k cuando se encuentra en el nodo i, es decir, el conjunto de nodos que la hormiga k aún no ha visitado.

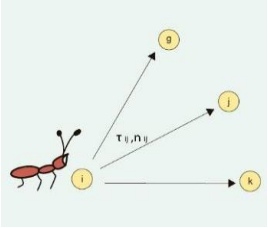

Figura 1. Una hormiga estando en el nodo i, a punto de elegir un nodo para visitar. Dorigo, M., Birattari, M., Stützle, T. (2006). Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique. [Figura].

2.2.2 Actualización de feromonas.

Una vez que todas las hormigas hayan creado su propia solución, es necesario efectuar el depósito de feromonas por los caminos que han recorrido (conexión de nodos) con su respectiva evaporación cumpliendo con los principios del algoritmo.

$$\tau_{ij}=(1-\rho)\tau_{ij} \quad (2)$$

donde $0 < \rho \leq 1$ es la velocidad de evaporación de feromona. El parámetro ρ se utiliza para evitar la acumulación ilimitada de los rastros de feromona y permite que el algoritmo "olvide" las malas decisiones tomadas previamente. De hecho, si las hormigas no eligen una arista, su valor de feromona asociado disminuye exponencialmente en el número de iteraciones. Después de la evaporación, todas las hormigas depositan feromona en las aristas que han cruzado en su recorrido (Dorigo & Stützle, Ant Colony Optimization, 2004):

$$\tau_{ij}= \tau_{ij}+ \sum_{k=1}^{m} \Delta\tau_{ij} \quad (3)$$

donde ΔT_{ij} es la cantidad de feromona depositada de la hormiga k en las aristas que ha visitado y se define de la siguiente manera:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k} , & \text{si la arista(i,j) pertenece a } T^k \\ 0, & \text{de lo contrario} \end{cases} \quad (4)$$

Donde C^k es la distancia del recorrido T^k construida por la k-ésima hormiga y se calcula como la suma de las longitudes de las aristas que pertenecen a T^k.

La hormiga que contiene la mejor solución es aquella que ha recorrido la menor distancia. Es importante tomar en cuenta que la distancia recorrida puede tomar diferentes interpretaciones según el problema, ya que depende mucho el valor heurístico que se haya definido; por ejemplo, en el problema del viajero, dicho recorrido viene siendo la distancia recorrida que se efectuó de ir viajando de una ciudad a otra hasta recorrer todas una sola vez y llegar a la misma ciudad de donde inició.

Algoritmo 1. Optimización Colonia de Hormigas (Dorigo, Birattari, & Stützle, Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique, 2006)

Establecer parámetros, inicializar caminos de feromonas
mientras condición de terminación no se haya cumplido **hacer**
Construir Soluciones de Hormigas
Aplicar búsqueda local(opcional)
Actualizar Feromonas
Fin mientras

El pseudocódigo describe de manera general cómo el algoritmo de optimización trabaja para poder encontrar una solución factible a un problema determinado. Se inicia cargando los parámetros (cantidad de hormigas, número de iteraciones, valores de alfa, beta) e inicializando el camino de feromonas. Se mantendrá en ejecución mientras que el criterio de finalización no haya sido cumplido (cantidad de iteraciones). Una hormiga construye una solución al problema y de manera opcional es posible aplicar el método búsqueda local para mejorar la solución. Posteriormente la hormiga realiza el depósito de feromonas. Al final, se obtiene una solución factible.

2.3 Max-Min Ant System.

El método que se ha elegido para resolver el problema de asignación de horarios es una variante del algoritmo ACO llamado Max-Min Ant System (MMAS siglas en inglés) (Dorigo & Stützle, Ant Colony Optimization, 2004). Cuenta con 4 diferencias en comparación con Ant System que fue el primer algoritmo de optimización de colonia de hormigas en resolver problemas de optimización combinatoria. Esas diferencias son:

1. Solamente la mejor hormiga local o la mejor hormiga global es quien puede hacer el depósito de feromonas.

$$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}^{best} \quad (5)$$

donde $\Delta T_{ij}^{best} = 1/C^{best}$. La hormiga que tiene permitido hacer el depósito de feromonas puede ser la mejor hormiga encontrada desde la ejecución del algoritmo (mejor hormiga global), en ese caso $\Delta T_{ij}^{best} = 1/C^{bs}$ o la mejor hormiga de la iteración actual, en ese caso $\Delta T_{ij}^{best} = 1/C^{ib}$, donde C^{ib} es la duración del mejor recorrido de la iteración.

2. Limita el posible rango de valores de rastro de feromonas al intervalo.

$$\tau_{(e,t)} \begin{cases} \tau_{min} \text{if } \tau_{(e,t)} < \tau_{min} \\ \tau_{max} \text{ if } \tau_{(e,t)} > \tau_{max} \\ \tau_{(e,t)} \text{ de lo contrario} \end{cases} \quad (6)$$

3. Los rastros de feromona se inicializan en el límite superior del rastro de feromona, junto con una pequeña tasa de evaporación de feromona, esto se define: $\frac{1}{\rho}$.
4. Los rastros de feromonas se reinician cada vez que el sistema se acerca al estancamiento o cuando no se ha generado un recorrido mejorado para un cierto número de iteraciones consecutivas.

2.4 Descripción del problema.

El problema que se abordó es el problema de asignación de horarios para un foro de propuestas de proyectos de investigación. El problema consiste en programar una serie de presentaciones de proyectos (eventos), en las que cada evento debe ser ubicado en un espacio de tiempo y un aula disponible en una o más fechas determinadas. El nivel de complejidad del problema depende de dos aspectos:

- Cada uno de los proyectos debe ser presentado ante un jurado conformado por 3 docentes, por lo que al asignar más de un proyecto en un mismo espacio de tiempo en diferentes aulas, los docentes que conforman el jurado deben ser diferentes, ya que ningún maestro debe estar en más de un aula en un mismo espacio de tiempo (ya que representaría un empalme de horas).
- Cada docente tiene su propio horario de trabajo dentro de la institución (horario disponible), por lo que se pretende que los eventos sean programados en un espacio de tiempo en donde los docentes que conforman el jurado para un evento específico, tengan la misma disponibilidad en el horario asignado para dicho evento.

2.5 Interpretación del problema.

Para poder llevar acabo el uso del algoritmo de optimización colonia de hormigas al problema de asignación de horarios, fue necesario analizar y formular el problema, además de identificar los factores que intervenían en el proceso. Se determina que el problema está formulado por cuatro tuplas (E, T, J, A) en donde:

- E: Es el conjunto de eventos $E = \{e_1, e_2, e_3, \ldots, e_n\}$ que se deben presentar.
- T: Es el conjunto de espacios de tiempo $T = \{t_1, t_2, t_3, \ldots, t_n\}$ que representa el periodo disponible donde un evento puede ser asignado. Por ejemplo el espacio de tiempo t_1 puede representar el periodo: 9:00 am – 10:00 am de un día Lunes.
- J: Es el conjunto de jurados $J = \{j_1, j_2, j_3, \ldots, j_n\}$ que representa a los docentes que evaluarán el evento $j_1 = \{m_1, m_2, m_3, \ldots, m_n\}$.
- A: Es el conjunto de aulas disponibles $A = \{a_1, a_2, a_3, \ldots, a_n\}$ en donde un evento podría ser asignado para poderse presentar.

Y sus factores (restricciones) son:

- Un docente no puede ser asignado en un mismo espacio de tiempo en dos aulas diferentes (restricción dura).
- Un evento no puede ser asignado en más de un espacio de tiempo (restricción dura).

- La cantidad de tiempo disponible del docente, debe ser compatible con la cantidad de espacios de tiempo requerida para los proyectos que debe evaluar (restricción dura).
- Un evento debe tener al menos un espacio de tiempo en común entre los docentes que conforman el jurado (restricción dura).
- Programar un espacio de tiempo de receso, en el que no debe programarse ningún evento (restricción suave).
- Los docentes deben ser asignados a un espacio de tiempo que se encuentre dentro del horario disponible (restricción suave).

Posteriormente en base en lo anterior se empieza a analizar la fórmula del algoritmo, para poder dar una interpretación sobre cada elemento que lo conforma y relacionarlo con alguna de las tuplas identificadas anteriormente. Concluido eso se obtiene que:

- P_{ij}^{k} = Probabilidad de elegir un espacio de tiempo j para el evento i.
- T_{ij} = Cantidad de feromona en la arista que une el evento i con el espacio de tiempo j.
- d_{ij} = Cantidad de violaciones suaves entre el evento i con el espacio de tiempo j.
- N_{i}^{k} = Espacios de tiempo disponibles para la hormiga k.

2.6 Representación de solución.

Cada vez que una hormiga asigna un evento a un espacio de tiempo, éste es almacenado dentro de un vector en donde la posición del elemento (índice) representa la posición del evento almacenado en otro vector que contiene todos los eventos que se desean programar. Los eventos están ordenados de manera ascendente tomando en consideración la cantidad de espacios en común que existe entre los docentes que conforman el jurado.

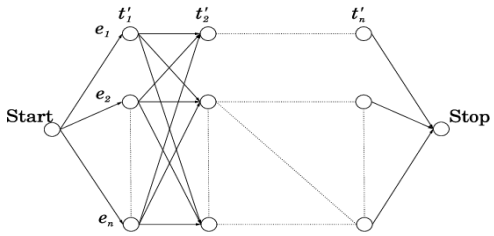

Figura 2. Cada hormiga sigue una lista de eventos, y para cada evento e Є E, una hormiga elige un espacio de tiempo t Є T. Rubio, J., Jonhson, F.,Crawford, B. (2008). ACO Hypercube Framework for Solving a University Course Timetabling Problem. [Figura].

En este problema específico la elección del aula de exposición es independiente de las características propias del aula, ya que se asume que cualquiera de ellas cumple con los requerimientos necesarios para el evento. Por lo tanto, un mismo espacio de tiempo puede ser elegido la misma cantidad de veces como la cantidad aulas disponibles, por lo que los espacios de tiempo por la cantidad de aulas (T X A) debe ser mayor o igual que la cantidad de proyectos a participar.

La solución del problema se almacena dentro de una matriz bidimensional M de tamaño T X A (T es el conjunto de espacios de tiempo y A es el conjunto de aulas disponibles), en donde una entrada en $M_{(i,j)}$ representa un evento asignado a un espacio de tiempo i en la sala j.

Tabla 1. Representación de la matriz M.

ET/Aulas	1	2
1	e_5	e_6
2		
3		e_8
4		

Algoritmo 2. Max-Min Ant System para resolver el problema de asignación de horarios (Rubio, Johnson, & Crawford, 2008)

Entrada: Cargar instancia I del problema
$\tau_{max} = \frac{1}{\rho}$
$\tau_{(e,t)} = \tau_{max}\ \forall\ (e,t) \in E \times T$
$E_{sorted} = sort(E)$
mientras condición de terminación no se haya cumplido **hacer**
    para k = 1 hasta m **hacer**
        A = Ø
        para i = 1 hasta |E| **hacer**
            elegir un espacio de tiempo **t** usando la influencia de la feromona y la información heurística para el evento $\mathbf{e_i}$ en E_{sorted}
            A {(ei, t)}
        fin para
        C = matching_algorithm (A_n)
        $C_{best\_iteration}$ = best C and $C_{best\_iteration}$
    end for
    $C_{best\_iteration}$ = aplicando búsqueda local a $C_{best\_iteration}$
    C_{global_best} = el major de $C_{best\_iteration}$ y C_{global_best}
    Actualización global de feromonas para τ usando C_{global_best}, τ_{max} y τ_{min}
fin mientras
Salida: Una solución candidata optimizada C_{global_best} for I

En el pseudocódigo se representan los pasos necesarios para la implementación del algoritmo Max-Min Ant System para dar solución al problema planteado en la sección 2.3. Al comienzo del proceso se cargan las instancias del problema y se inicializan los parámetros que el algoritmo necesita (τ_{max}, τ_{min}, la matriz de feromonas). Posteriormente los proyectos son ordenados de manera ascendente tomando como criterio la cantidad de espacios de tiempo en común que tienen los docentes que conforman el jurado en cada uno de los proyectos. Se mantendrá en ejecución mientras que el criterio de finalización aún no se haya cumplido; todas las hormigas crean una solución por cada iteración y se elige la mejor tomando como criterio la que menor violación de restricciones haya hecho; la hormiga con la mejor solución se le aplica búsqueda local para quitar todas las restricciones duras y suaves que se tengan (las restricciones duras deben eliminarse por completo, caso contrario el proceso se termina), se hace una comparación entre la solución de la mejor hormiga local y la mejor hormiga global para determinar cuál es la hormiga con la mejor solución desde la ejecución del algoritmo para realizar el depósito de feromonas. Al final de la ejecución, se obtiene la solución factible, proveniente por la mejor hormiga global.

3. Desarrollo.

Para poder comprobar que este problema de asignación de horarios puede ser resuelto a través de este método metaheurístico, fue necesario desarrollar una aplicación en Java y simular un foro.

Como parte de la simulación del foro, se establecieron dos fechas para presentar todos los proyectos en un horario de 8:00am a 2:00 pm (horario para ambas fechas). La cantidad de proyectos participantes será de 37; la cantidad de docentes participantes será de 22 y a cada maestro se le asigna las horas que tendrá disponible para participar en el foro, a partir de los 22 docentes participantes, se empiezan a crear los jurados para posteriormente asignarlos a un proyecto. Cada proyecto tiene como límite de tiempo 30 minutos. Por lo tanto la cantidad de espacios de tiempo es de 24 (existe una diferencia de 6 horas entre la hora de inicio y la hora final, multiplicado por 2 que es el resultado de dividir 60 minutos (cantidad de minutos en una hora) entre el límite de tiempo para presentar el proyecto, en este caso 30 minutos, multiplicado por la cantidad de fechas).

Tabla 2. Instancias del problema

Parámetros	Instancias
Proyectos	37
Espacios de tiempo	24
Maestros	22
Aulas	2

3.1 Implementación del algoritmo.

La figura 3 muestra el diagrama de clases del programa desarrollado para este fin, en notación UML. A continuación se explica cada una de las clases y de los métodos relevantes.

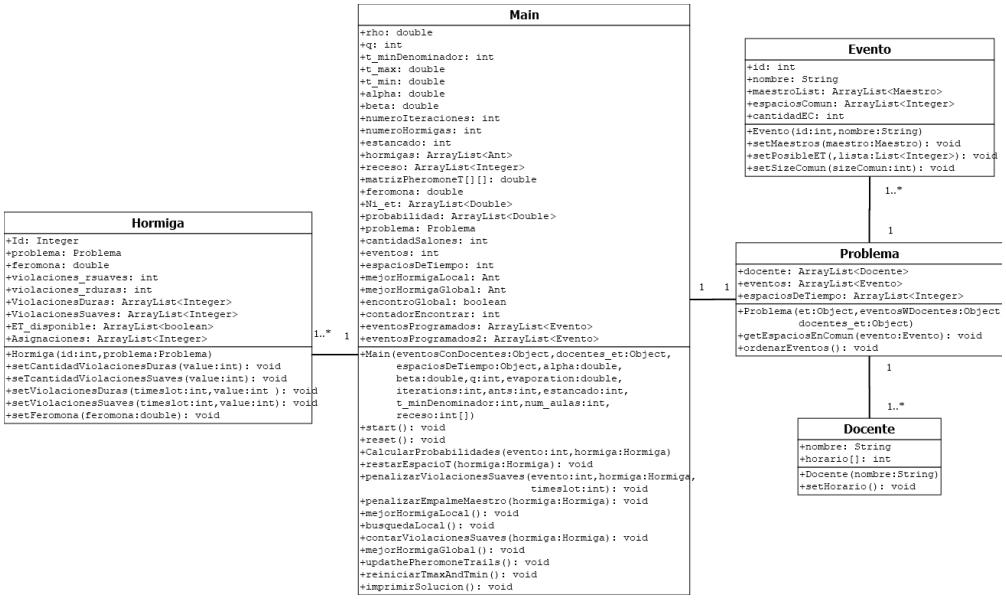

Figura 3. Diagrama UML de las clases utilizadas para el desarrollo del proyecto en Java.

3.1.1 Hormiga.

Cada instancia de la clase hormiga, representa las hormigas artificiales del algoritmo, son las que crearán una solución al problema de horarios.

3.1.1.1 Métodos.

- *setCantidadViolacionesDuras()*: Asigna un valor para el atributo “violaciones_rsuaves”. Representa la cantidad total de violación de restricciones duras existentes en todos los espacios de tiempo.
- *setCantidadViolacionesSuaves()*: Asigna un valor para el atributo “violaciones_rduras”. Representa la cantidad total de violación de restricciones suaves existentes en todos los espacios de tiempo.
- *setViolacionesDuras()*: Penaliza con +1 el espacio de tiempo elegido para un evento en caso de violar una restricción dura.
- *setViolacionesSuaves()*: Penaliza con +1 el espacio de tiempo elegido para un evento en caso de violar una restricción suave.

- *setFeromona()*: Asigna al atributo “feromona” la cantidad de feromona que una hormiga depositará en las aristas que ha visitado.

3.1.2 Evento.

Clase en donde cada instancia representa a un proyecto al cual se le necesita asignar un espacio de tiempo y un aula.

3.1.2.1 Métodos.

- *setMaestros()*: Agrega al atributo “docenteList” los docentes que conformaran el jurado para cada proyecto.
- *setPosiblesEspaciosT()*: Agrega al atributo “espaciosComun” los espacios de tiempo que tienen en común los docentes que conforman el jurado de cada proyecto.
- *setSizeComun()***:** Asigna al atributo “cantidadEC” la cantidad de espacios de tiempo en común que tiene el jurado en cada proyecto.

3.1.3 Docente.

Clase en donde cada instancia representa a un maestro, quien podría estar involucrado en al menos un proyecto.

3.1.3.1 Métodos.

- *setHorario()*: Agrega al atributo “horarios” los espacios de tiempo disponible de cada docente.

3.1.4 Problema.

Clase que represente el problema de horarios, contiene instancias de la clase evento, la clase maestro. Además contiene las fechas y las horas donde los eventos podrán ser asignados.

3.1.4.1 Métodos.

- *getEspaciosEnComun()*: Selecciona cuales son los espacios de tiempo en común que tienen los docentes que conforman el jurado en cada proyecto.
- *ordenarEventos()*: Ordena los eventos que están almacenados en la variable “eventos” de manera ascendente considerando el atributo “cantidadED” de la clase evento.

3.1.5 Main.

Clase principal que contiene la lógica del algoritmo Max-Min Ant System y otras funciones que fueron fundamentales para el buen funcionamiento del algoritmo.

3.1.5.1 Métodos.

- *Start()*: Inicializa el programa.
- *Reset()*: Reinicia los atributos de “probabilidad” y “ET_disponible” de la clase hormiga, para olvidar los cálculos generados en las iteraciones anteriores.
- *CalcularProbabilidades()*: Calcula las probabilidades de elegir un espacio de tiempo para un proyecto.
- *restarEspacioT()*: Cuenta la cantidad de veces que ha sido elegido un espacio de tiempo determinado durante la ejecución. Si la cantidad de veces es igual a la cantidad de aulas disponibles, cambia el estado de falso a verdadero en el atributo “ET_disponible” del espacio de tiempo elegido, indicando que en ese espacio de tiempo ya no hay aulas disponibles; por lo tanto no podrá ser elegido nuevamente.
- *penalizarViolacionesSuaves()*: Penaliza con +1 si el espacio de tiempo elegido para un evento viola alguna restricción suave.
- *penalizarEmpalmeMaestro()*; Penaliza con +1 si los proyectos asignados a un mismo espacio de tiempo genera un empalme de horas entre los maestros involucrados.

- *mejorHormigaLocal()*: Elige la hormiga con la mejor solución tomando en consideración la cantidad de violación de restricciones suaves.
- *busquedaLocal()*: Mejora la solución de la mejor hormiga local disminuyendo las violaciones de restricciones suaves y duras.
- *contarViolacionesSuaves()*: Cuenta la cantidad de violación de restricciones suaves generadas en la solución.
- *mejorHormigaGlobal()*: Compara dos hormigas, la hormiga con la mejor solución de la iteración actual y la hormiga con la mejor solución creada desde la ejecución del programa, la hormiga que tenga la solución con menor cantidad de violación de restricciones suaves será definida como mejor hormiga global.
- *updatePheromoneTrails()*: La hormiga declarada como mejor hormiga global, deposita la cantidad de feromonas por las aristas que ha recorrido durante la construcción de solución.
- *reiniciarTmaxAndTmin()*: Reinicia los valores de la matriz de feromonas tomando en consideración la formula númro (6) descrita en la sección 2.3.
- *imprimirSolucion()*: Muestra la solución construida por la mejor hormiga global, almacenada en la matriz bidimensional M.

3.2 Pruebas.

Se realizaron diferentes pruebas con las mismas instancias del problema, pero con variaciones en los parámetros del algoritmo. Los resultados son mostrados en la siguiente tabla.

Tabla 3. Resultados obtenidos de las pruebas con diferentes valores en algunos parámetros.

Prueba	α	β	Hormigas	ρ	Iteraciones	Violaciones suaves	Tiempo de ejecución (seg)
1	1	2	5	0.1	100	3	2
2	1	2	5	0.1	150	1	3
3	1	2	5	0.1	200	0	4
4	0.1	2	5	0.1	200	8	5
5	0.8	2	5	0.1	200	2	4
6	0.5	2	5	0.9	200	4	4
7	0.1	1	10	0.1	1000	5	59

Conclusiones.

A través de las pruebas realizadas y los resultados obtenidos se concluye que es posible resolver un problema de asignación de horarios a través de la variante Max-Min Ant System del algoritmo de optimización colonia de hormigas junto al método de búsqueda local. Sin embargo la calidad de la solución depende demasiado de los valores de algunos parámetros que se establecen, del tamaño del problema y algunos valores que pertenecen a las instancias del problema (por ejemplo la cantidad de espacios en común que tienen los docentes que conforman el jurado de un proyecto). Se observó que la solución factible al problema puede ser encontrada con un menor número de iteraciones si $\alpha = 1$ y $\rho = 0.1$; si un proyecto solo tiene un espacio en común, será necesario incrementar el número de iteraciones.

Referencias Bibliográficas.

Dorigo, M., Birattari, M., & Stützle, T. (Noviembre de 2006). Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique. *IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE*, 28-39.

Dorigo, M., & Blum, C. (Noviembre de 2005). Ant colony optimization theory: A survey. *Theoretical Computer Science, 344*, 243 – 278.

Dorigo, M., & Stützle, T. (2004). Ant Colony Optimization. Cambridge, Massachusetts, Estados Unidos: MIT Press.

Peñuela, C. A., Toro, E., & Franco, J. F. (Junio de 2008). Colonia de hormigas aplicada a la programación óptima de horarios de clase. *Scientia et Technica*(38), 49-54.

Rubio, J. M., Johnson, F., & Crawford, B. (2008). ACO Hypercube Framework for Solving a University Course Timetabling Problem. *Proceedings of the International MultiConference of Engineers and Computer Scientists, I.* Hong Kong.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review 13, 13*, 87–127.

Socha, K., Sampels, M., & Manfrin, M. (s.f.). Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art.

Información de los autores.

Oswaldo de Jesús Chacón Pérez, es alumno del Instituto Tecnológico de Tuxtla Gutiérrez de la carrera de Ingeniería en Sistemas Computacionales y se especializa en el desarrollo de aplicaciones móviles y tecnologías web. Sus principales intereses son: desarrollo de aplicaciones web y aplicaciones android.

Jennifer Vanessa Aguilar Padilla, es alumna del Instituto Tecnológico de Tuxtla Gutiérrez de la carrera de Ingeniería en Sistemas Computacionales y se especializa en el desarrollo de aplicaciones móviles y tecnologías web.

Octavio Ariosto Ríos Tercero, Maestro en Ciencias en Ciencias Computacionales egresado del Centro Nacional de Investigación y Desarrollo Tecnológico CENIDET. Su experiencia en docencia es en el área de ingeniería de software. Profesor de tiempo completo con perfil Promep, actual Jefe de Proyectos de Investigación en el Departamento de Sistemas y Computación del Tecnológico Nacional de México campus Tuxtla Gutiérrez.

Rosy Ilda Basave Torres. Maestra en Ciencias en Ciencias de la Computación egresada del Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET). Es profesora en el área de Ingeniería en Sistemas Computacionales del Instituto Tecnológico de Tuxtla Gutiérrez, es miembro del cuerpo académico ITTUXG-CA-7, colabora en la línea de investigación ITFTGTZ-LIE-2018-0163, tiene el reconocimiento al perfil deseable y es miembro del Sistema Estatal de Investigadores.

Roberto Cruz Gordillo es Ingeniero Electrónico por el Instituto Tecnológico de Tuxtla Gutiérrez en 1996 y Maestro en Ciencias en Diseño Asistido por Computadora (CAD) por el Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE) en 2003, actualmente es profesor de medio tiempo del Instituto Tecnológico de Tuxtla Gutiérrez desde enero de 2004, asignado al Departamento de Sistemas y Computación, se especializa en matemáticas.