PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# RESOLUCIÓN DEL PROBLEMA DE BALANCEO DE MALLAS CURRICULARES UTILIZANDO OPTIMIZACIÓN BASADA EN COLONIA DE HORMIGAS

**DIEGO ARNALDO ARAVENA DIAZ**

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TITULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

ENERO 2012

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

# RESOLUCIÓN DEL PROBLEMA DE BALANCEO DE MALLAS CURRICULARES UTILIZANDO OPTIMIZACIÓN BASADA EN COLONIA DE HORMIGAS

**DIEGO ARNALDO ARAVENA DIAZ**

Profesor Guía: **José Rubio León**

Carrera: **Ingeniería Civil Informática**

Enero 2012

# Dedicatoria

A mis padres por todo el cariño, sacrificio e incondicional apoyo a lo largo de toda mi vida.
A mis hermanos por darme ánimo y siempre creer en mí.

# Índice

# Lista de Abreviaturas o Siglas

BACP: Balanced Academic Curriculum Problem.

OCH: Optimización basada en Colonia de Hormigas.

SH: Sistema de Hormigas.

SCH: Sistema de Colonia de Hormigas.

SHMM: Sistema de Hormigas Max-Min.

SMPH: Sistema de la Mejor-Peor Hormiga.

TSP: Problema del Vendedor Viajero (del Ingles Travelling Salesman Problem).

CSP: Problema de satisfacción de restricciones.

# Lista de Figuras

# Lista de Tablas

# Resumen

El problema de Balanceo de Mallas Curriculares consiste en la asignación de cursos a periodos académicos con el fin de que la carga académica se encuentre balanceada, es decir, que todos los semestres tengan un número similar de créditos y que la mayor carga académica entre todos los periodos sea la menor posible, respetando ciertas restricciones impuestas por las diversas casas de estudio. Para diseñar una malla balanceada existen diversas técnicas dentro del área de optimización. En este trabajo se resolvió este problema por medio de la Optimización basada en Colonia de Hormigas.

Las hormigas artificiales son agentes computacionales que se encargan de construir las soluciones por medio de un grafo de construcción tomando decisiones basadas en los rastros de feromona y la información heurística entre cada arista. Se decidió trabajar con el Sistema de la Mejor-Peor Hormiga que agrega 3 elementos que lo convierten en una buena alternativa para la solución de este problema: efectúa una deposición y evaporación de feromona adicional usando las hormigas que generen la mejor y la peor solución, aplica mutación en los rastros de feromona para dar diversidad y así explorar mejor el espacio de búsqueda de las soluciones y considera una reinicialización de los rastros de feromona en el caso que se estanque la búsqueda en un óptimo local.

En el presente trabajo se resolvieron tanto instancias de prueba como instancias reales con el fin de medir el rendimiento del algoritmo y entregar soluciones que permitan simplificar el proceso de diseño de una malla curricular.


**Palabras clave:** BACP, Optimización basada en Colonia de Hormigas, Búsqueda local, Metaheurísticas.



# Abstract

The Balanced Academic Curriculum Problem consists in the assignation of courses to academic periods with the purpose of the academic load is balanced, i.e., all the periods have a similar amount of credits and the higher academic load between all the periods be the smallest possible, respecting certain constraints imposed by the different school houses. To design a balanced curricula there are a lot of techniques inside the optimization field. This work set out to solve this problem using Ant Colony Optimization.

Artificial Ants are computational agents that are responsible of build the solutions through a construction graph and making decisions based in the pheromone trails and heuristic information between edges. It was decided to work with the Best-Worst Ant System that adds three new elements that make it a good alternative for solving the problem: performs an extra pheromone deposition and evaporation using the ants that generates the best and worst solutions, apply mutation in the pheromone trails to bring diversity and thus better explore the search space of the solutions and considers a reset of the pheromone trails in case the search of the local optimal stagnates.

In this work it was solved both, real instances as test instances to measure the performance of the algorithm and deliver solutions that enables to simplify the process of designing an academic curricula.


**Key words:** BACP, Ant Colony Optimization, Local Search, Metaheuristics.

# 1 Introducción

Cuando se diseñan mallas curriculares para una carrera Universitaria se toman en consideración factores como la cantidad de asignaturas que posee la carrera, el número de periodos en el que se deben asignar estos cursos, la carga académica mínima y máxima aceptada, y el número de cursos permitido por semestre. Con toda esta información se conforman las restricciones o regulaciones académicas relacionadas al plan de estudios, las cuales son usadas para su construcción mediante un enfoque de prueba y error hasta lograr una malla adecuada.

Ya que el grado de esfuerzo que se requiere para aprobar una asignatura se puede medir en créditos, el éxito que puedan tener los estudiantes tiene directa relación con la carga académica que enfrenten en cada periodo. La carga académica corresponde al número de créditos por semestre. Es por esta razón que es de importancia que las mallas curriculares se encuentren balanceadas, es decir, la cantidad de créditos de cada periodo sea similar, para que la carga que enfrenten los alumnos sea la menor posible. Por lo tanto, es de interés minimizar este costo incurrido en diseñar un plan de estudios mediante un algoritmo que realice este esfuerzo de manera automática y sin incurrir a errores.

Este problema es conocido en la literatura como el problema de balanceo de mallas curriculares (BACP), y es del tipo CSP (Problema de Satisfacción de Restricciones), donde primero se busca satisfacer todas las restricciones asociadas y luego optimizar la calidad de la solución encontrada. Existen prototipos y modelos estudiados que resuelven el BACP. Generalmente ha sido abordado utilizando el paradigma de programación con restricciones y algoritmos híbridos donde se utilizaron algoritmos genéticos, esquemas de colaboración, búsquedas locales, etc.

Este trabajo de investigación se centró en resolver tanto las instancias de prueba reconocidas por la CSPLib[1] como instancias reales. Entre estas se encuentran las dos carreras de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso más la carrera de Ingeniería Informática de la Universidad de Playa Ancha. Se utilizó la metaheurística de optimización basada en colonia de hormigas, la cual consiste en emular el comportamiento de las hormigas naturales para encontrar fuentes de alimentos desde su nido por medio de la ruta más corta. Para lograrlo, estos insectos depositan una sustancia química llamada feromona la cual guiará a los siguientes miembros de la colonia.

La optimización basada en Colonia de Hormigas es capaz de resolver problemas de satisfacción de restricciones, un ejemplo es el problema del Timetabling, que consiste en asignar los horarios de clases a un cierto número de recursos humanos, es decir, asignar las clases a los periodos del día y a luego a una sala de clases. Bajo esta premisa se puede mapear el problema del Timetabling al problema de balanceo de mallas curriculares. Mientras que

[1] Librería con gran cantidad de problemas de satisfacción de restricciones. Disponible en http://www.csplib.org/

otros estudios se concentran en la satisfacción de restricciones, en este trabajo además se invierten esfuerzos en optimizar la calidad de las soluciones encontradas.

## 1.1 Estructura del trabajo de investigación

En el Capítulo 2 de este trabajo, se presentó el problema del balanceo de mallas curriculares y las restricciones que deben tomarse en cuenta para cumplir con las regulaciones administrativas y académicas de las instituciones. En Capítulo 3 se presentó la investigación relacionada al problema del balanceo de mallas curriculares, quienes lo han resuelto y que heurísticas o técnicas se han utilizado. En el Capitulo 4 se presentó, con el fin de lograr una mayor comprensión del proyecto, los conceptos más relevantes respecto al problema y la técnica propuesta para resolverlo, que en este caso viene ser la optimización basado en colonia de hormigas. En el Capítulo 5 se mostró la propuesta respecto de cómo resolver el problema con la heurística seleccionada, esto mediante pseudocódigos y las formulas a utilizar. En el capítulo 6 se describió el prototipo realizado y un análisis de los resultados arrojados por este. Finalmente, las conclusiones respecto a la investigación realizada y el trabajo futuro que se desprende de éste.

## 1.2 Objetivos

En la siguiente sección se presentó el objetivo general y los objetivos específicos de este trabajo de titulo.

### 1.2.1 Objetivo General

- Resolver el problema de balanceo de mallas curriculares mediante Optimización basada en colonia de hormigas.

### 1.2.2 Objetivos Específicos

- Desarrollar un modelo de balanceo de mallas curriculares basada en optimización con colonia de hormigas.
- Diseñar un esquema de solución del problema de balanceo de mallas curriculares.
- Implementar un algoritmo de balanceo de mallas curriculares utilizando optimización basada en colonia de hormigas.
- Evaluar el rendimiento del algoritmo propuesto mediante la utilización de instancias benchmarks y reales.

# 2 Definición del problema

El problema de Balanceo de mallas curriculares (BACP) busca asignar a los periodos cursos, de manera que la carga académica de cada semestre este balanceada, esto quiere decir que cada periodo debe tener una cantidad similar de créditos. Además, la malla debe respetar las siguientes restricciones [1]:

- El currículo académico debe contener un conjunto de cursos y prerrequisitos relacionados a estos.
- Los cursos deben ser asignados dentro de un número máximo de periodos académicos.
- Cada curso tiene asociado un número de créditos o unidades que representa el esfuerzo académico requerido de este.
- Varios cursos necesitan la aprobación de otros para poder ser cursados (prerrequisitos).
- Se debe cumplir una cantidad mínima y máxima de créditos por cada periodo académico.

Se busca asignar un periodo a cada curso para satisfacer las restricciones de número mínimo y máximo de créditos por periodos y además se cumplan todas las prerrequisitos. Un currículo académico balanceado óptimo minimiza la máxima carga académica para cada periodo.

## 2.1 Modelo matemático

El modelo de programación lineal entero propuesto para resolver el problema de balanceo de mallas curriculares fue el usado por Carlos Castro y Sebastián Manzano en [2] quienes introdujeron este problema:

- Parámetros:
  - m: Número de cursos
  - n: Número de periodos académicos
  - $\alpha_i$ : Número de créditos de curso i=1,…,m
  - β: Carga académica mínima por periodo
  - γ: Carga académica máxima por periodo
  - δ: Cantidad mínima de cursos por periodo
  - ε: Cantidad máxima de cursos por periodo

- Variables de decisión:
  - $x_{ij} = \begin{cases} 1 \; si \; el \; curso \; i \; es \; asignado \; al \; periodo \; j \\ 0 \; en \; otro \; caso \end{cases} \quad \forall i = 1,...m \; \forall j = 1,..., n$
  - c: la máxima carga académica para todos los periodos c = Max{$c_1$,…,$c_n$}
  - La carga académica del periodo j está definida por

    $$c_j = \sum_{i=1}^{m} \alpha_i x_{ij}, \; \forall i = 1,..., m$$

- Función Objetivo:
  - Min c
- Restricciones:
  - Todos los cursos i deben ser asignados a un periodo j:
    $$\sum_{i=1}^{n} x_{ij} = 1, \forall i = 1,...,m$$
  - El curso b tiene un curso a como prerrequisito:
    $$x_{bj} \leq \sum_{r=1}^{j-1} x_{ar}, \forall j = 2,...,n$$
  - La carga académica máxima está definida por: c = Max $\{c_1,\ldots, c_n\}$. Lo que puede ser representado por el siguiente conjunto de restricciones lineales:
    $$c_j \leq c \; \forall j = 1,\ldots,n$$
  - La carga académica del periodo j debe ser mayor o igual que el mínimo requerido:
    $$c_j \geq \beta \; \forall j = 1,\ldots,n$$
  - La carga académica del periodo j debe ser menor o igual que el máximo requerido:
    $$c_j \leq \gamma \; \forall j = 1,\ldots,n$$
  - El número de cursos del periodo j debe ser mayor o igual que el mínimo requerido:
    $$\sum_{i=1}^{m} x_{ij} \geq \delta \; \forall j = 1,\ldots,n$$
  - El número de cursos del periodo j debe ser menor o igual que el máximo requerido:
    $$\sum_{i=1}^{m} x_{ij} \leq \varepsilon \; \forall j = 1,\ldots,n$$

# 3 Estado del arte

En esta sección se muestran los autores y las técnicas o paradigmas utilizados para resolver o modelar el problema de balanceo de mallas curriculares desde su invención hasta el día de hoy.

El problema de balanceo de mallas curriculares (BACP) fue introducido por Castro y Manzano en el año 2001 en [2], y era parte de la CSPLib (Gent and Walsh1999, problema 30) con tres instancias de prueba. En su formulación original, el balance de carga académico se lograba minimizando la máxima carga académica por estudiante. En ese trabajo, Castro y Manzano resolvieron el problema usando Programación con restricciones, demostrando que es una técnica más eficiente que la programación lineal entera para resolver problemas de optimización combinatorios.

El BACP ha sido abordado también usando programación con restricciones y programación lineal entera por Hnich, Kiziltan y Walsh en el año 2002 [11], donde probaron integrar los distintos modelos para así lograr complementar las fuerzas de cada modelo. Los resultados experimentales dejaron en claro que al integrar los modelos se incrementaba el dominio de la poda y disminuía el tiempo de ejecución de varias instancias.

Lambert, Castro, Monfroy y Saubion utilizaron técnicas hibridas compuesta por *Algoritmos Genéticos* y *Propagación de restricciones* en el año 2005 [12]. Ellos desarrollaron un modelo teórico en donde se logró una resolución hibrida. En estos trabajos, los autores fueron capaces de calcular la solución óptima para dos instancias pequeñas presentadas en la CSPLib en menos de 40 segundos. La tercera instancia aun no se resuelve pasada una hora.

Monette, Schaus, Zampelli, Deville y Dupont en el 2007 estudiaron el BACP extensamente. Ellos lograron resolver las tres instancias de la CSPLib de manera óptima, además, introduce un generador de instancias aleatorias para obtener más instancias desafiantes. Ellos concentraron sus experimentos en 720 instancias con 200 cursos y variadas características. Lo más importante es que ellos dedicaron su atención al criterio de balance de carga, extendiendo la carga máxima original sobre los periodos, usada en los trabajos previos, hasta la máxima desviación desde la carga promedio hasta la suma cuadrática y lineal de esas desviaciones.

En el año 2007, Carlos Castro y Broderick Crawford resolvieron modelos matemáticos utilizando técnicas completas e incompletas, para diseñar mallas curriculares balanceadas. Luego en el 2009, José Miguel Rubio junto a Crawford publicaron un trabajo donde modelaron y diseñaron un esquema de colaboración de programación con restricciones y técnicas de búsqueda local para la resolución del BACP utilizando propagación de restricciones y estrategias de búsqueda local inspirada en un algoritmo de mejora iterativa estocástica [1].

En el trabajo de Chiarandini, Di Gaspero, Gualandi y Schaerf, publicado recientemente (Enero 2011) y titulado *The balanced academic curriculum problem revisited*, ellos dejan en claro que el problema es más simple que el problema real que deben resolver las

Universidades en la práctica. Por lo tanto, propusieron superar esa limitación definiendo una nueva formulación más general que incluye la anterior y modelando la situación de una forma más real. Esta nueva formulación hace posible incluir más de un currículo, con cursos compartidos entre planes de estudios. Incluso, contiene las preferencias de los profesores y la indisponibilidad de enseñar en algunas ocasiones [10]. Ellos introdujeron 10 nuevas instancias obtenidas de la Universidad de Udine. Estas instancias eran mucho más grandes y exhibían estructuras distintas, dado que venía de casos diferentes. Finalmente esta formulación y las nuevas instancias resultaron ser más difíciles de resolver que las previas instancias. Por lo que no se pudo llegar a una solución óptima.

# 4 Marco Teórico

En el siguiente apartado se describieron los conceptos más relevantes para la comprensión del trabajo realizado. Los temas a abordar son relacionados a las mallas curriculares, las metaheurísticas, el área de optimización combinatoria, la optimización basada en colonia de hormigas y la búsqueda local.

## 4.1 Malla curricular en la enseñanza superior

Tradicionalmente se ha designado al plan de estudios como un conjunto de asignaturas y actividades graduadas, sistematizadas y armonizadas, de manera que concurran a la obtención de un objetivo o grupo de objetivos, correspondientes a un nivel educativo.

Dentro de los esquemas de la pedagogía moderna, el plan de estudios es denominado *Currículo*, y se le define como "El conjunto de enseñanzas, teorías y prácticas, que han de realizar para ser promovidos los alumnos, como el orden de ellos dentro de una institución docente" [3].

Todos los procedimientos modernos para organizar la materia de enseñanza aspiran a que el currículo sea orgánico y funcional. Un plan orgánico de enseñanza enlaza de un modo natural y múltiple las asignaturas o temas concretos, mediante una red de comunicaciones que permite aproximar los contenidos más diversos del saber y de la técnica, evitando la dispersión mental de los alumnos y logrando un efecto total. Como el verdadero aprender implica una transformación graduada y valiosa de las aptitudes humanas, el currículo orgánico concibe de peculiar modo las materias de enseñanza: éstas dejan de ser meros signos de erudición e información, y se convierten en medios eficientes para la realización de la vida presente y futura, de los aspirantes a profesionales.

La materia de enseñanza se selecciona y ordena para crear en el alumno la mejor habilidad en las situaciones de la profesión, y el aprendizaje queda así articulado en función del círculo de experiencias actuales y posibles del alumno. Los nuevos currículos de estudio son a un tiempo planes de formación y planes de materias, ofrecen cuadros de experiencias, métodos de trabajo y orientación para evaluar los resultados del aprendizaje. Indican en una unidad diferenciada qué materia puede y debe ser asimilada por el estudiante y cómo puede ello realizarse.

### 4.1.1 Organización de la malla curricular

Las asignaturas que se dictan en las instituciones de educación superior se pueden clasificar de la siguiente manera:

- **Cursos básicos a nivel general:** Corresponden a las asignaturas que proporcionan a los estudiantes una cultura básica universitaria en distintas áreas, como las ciencias o las humanidades. Estas materias no se dictan en función de ninguna especialidad, sino como núcleo y fundamento para la formación profesional.

- **Cursos requisitos del programa académico:** Son las asignaturas que deben ser estudiadas y aprobadas por todos los estudiantes que siguen un programa académico. Los cursos requisitos del programa académico son seleccionados, coordinados y evaluados por la dirección del programa académico respectivo, y constituyen el núcleo básico de cursos de la profesión o de la especialidad.

- **Cursos electivos:** Estos son las asignaturas en que el estudiante puede elegir entre varias posibilidades sin restricciones, pero siempre debe llevarlos para completar el número total de créditos que le exige el currículo de su programa académico.

### 4.1.2 El sistema de créditos

La flexibilidad de la malla curricular requiere un instrumento operativo que permita estimar el trabajo académico de los estudiantes y traducirlo en cifras que revelen en cualquier instante su situación y su progreso dentro de la universidad. Este instrumento es el sistema de créditos. El *crédito* es una unidad de evaluación del trabajo efectuado para aprobar una asignatura.

Expresando en créditos el volumen de trabajo que representa cada curso, se facilita el establecimiento de pautas que regulen la distribución equilibrada de asignaturas en la malla curricular y el monto de la carga académica ponderada del rendimiento del estudiante en las diversas materias.

Los créditos representan un procedimiento exacto y racional para dar significado a las calificaciones e interpretar en cifras la situación académica real de los estudiantes. El sistema permite establecer niveles mínimos de rendimiento y determinar cuando un alumno está encontrando dificultades en sus estudios y necesita mayor ayuda y superar sus dificultades. Un índice ponderado demasiado bajo es una señal de alarma sobre la situación académica del alumno y sus probabilidades de fracasar como estudiante.

## 4.2 Conceptos relacionados al área de Optimización

En esta sección se describieron algunos conceptos relacionados al área de optimización combinatoria con el fin de lograr una mayor comprensión de la naturaleza del problema y como se resolverá.

### 4.2.1 Optimización combinatoria

Es una rama de la optimización en matemáticas aplicadas y en ciencias de la computación, relacionada a la investigación de operaciones, teoría de algoritmos y teoría de la complejidad computacional. Los algoritmos de optimización combinatoria resuelven instancias de problemas que se creen ser difíciles en general, explorando el espacio de soluciones (usualmente grande) para estas instancias. Los algoritmos de optimización combinatoria logran esto reduciendo el tamaño efectivo del espacio, y explorando el espacio de búsqueda eficientemente.

#### 4.2.1.1 Instancia de un problema de optimización

Corresponde a un par (F, c), donde "F" es cualquier conjunto dentro del dominio de puntos factibles y "c" es la función de costos, tal que [7]:

$c : F \to \Re$

El problema es encontrar un $f \in F$, tal que:

$$c(f) \leq c(y), \forall y \in F \quad (1)$$

#### 4.2.1.2 Vecindario

Dado un punto factible $f \in F$ en una instancia, se define vecindario al conjunto de puntos N (f) que están "cerca" del punto f.

#### 4.2.1.3 Óptimo local

Dada una instancia (F, c) de un problema de optimización y un vecindario N, una solución f es llamada Óptimo local respecto de N si:

$$c(f) \leq c(g), \forall g \in N(f) \quad (2)$$

#### 4.2.1.4 Óptimo global

El óptimo global se puede definir como el mejor Óptimo local, de todos los vecindarios en una instancia de un problema de optimización.

Dada una instancia (F, c) de un problema de optimización y N (f)*, el conjunto de todos los vecindarios posibles, f será óptimo global si:

$$c(f) \leq c(y), \forall y \in N(f)^* \quad (3)$$

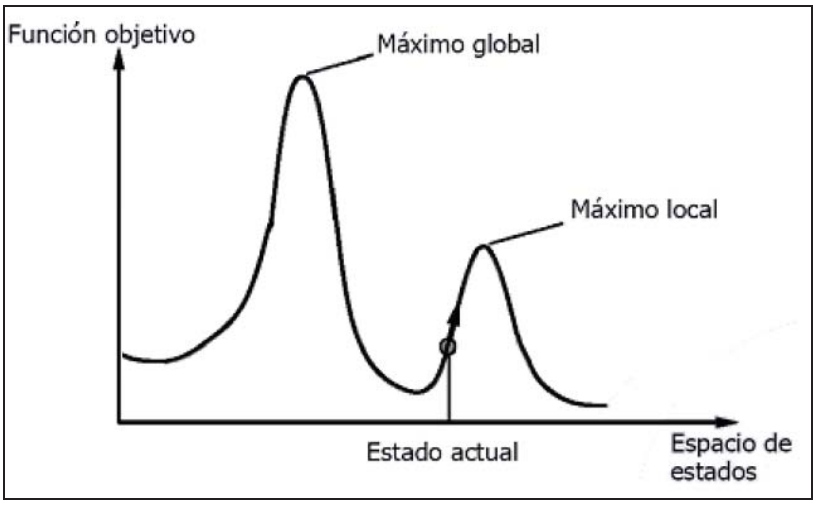


**Figura 4.1: Ejemplo de máximo global y máximo local**

### 4.2.2 Metaheurísticas

Clase de algoritmos aproximados[2], cuya idea básica es combinar diferentes métodos heurísticos a un nivel más alto para conseguir una exploración del espacio de búsqueda de forma eficiente y efectiva. Las metaheurísticas presentan las siguientes propiedades [8]:

- Las metaheurísticas son estrategias o plantillas generales que guían el proceso de búsqueda.
- El objetivo es una exploración eficiente del espacio de búsqueda para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son algoritmos no exactos y generalmente son no deterministas[3].
- Pueden incorporar mecanismos para evitar regiones no prometedoras del espacio de búsqueda.
- El esquema básico de cualquier metaheurística tiene una estructura predefinida.
- Las metaheurísticas pueden hacer uso de conocimiento del problema que se trata de resolver en forma de heurísticos específicos que son controlados por la estrategia de más alto nivel.

Resumiendo estos puntos, se puede acordar que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda.

#### 4.2.2.1 Clasificación de las Metaheurísticas

Las metaheurísticas se clasifican en dos grupos: las metaheurísticas *basadas en trayectoria* y las metaheurísticas *basadas en población*. Las primeras manipulan un elemento en el espacio de búsqueda, mientras que las segundas trabajan sobre un conjunto de elementos (población).

[2] Algoritmos que no garantizan que la solución encontrada sea el óptimo global.
[3] Tipos de algoritmo que dada una entrada, no siempre producen la misma salida.

Entre las basadas en trayectoria encontramos:

- Enfriamiento simulado (SA)
- Búsqueda Tabú (TS)
- GRASP
- Búsqueda con vecindario variable (VNS)
- Búsqueda local iterada (ILS)

Entre las basadas en población encontramos:

- Algoritmos evolutivos (EA)
- Algoritmos de estimación de la distribución (EDA)
- Búsqueda dispersa (SS)
- Optimización por enjambre de partículas (PSO)
- Optimización basada en Colonia de Hormigas (OCH)

En la siguiente sección se puso énfasis únicamente en esta última metaheurística, la optimización basada en Colonia de Hormigas.

## 4.3 Optimización basada en colonia de hormigas

La optimización basada en colonia de hormigas es una metaheurística relativamente reciente que se inspira en el comportamiento de las hormigas naturales para encontrar el camino más corto entre las fuentes de comida y el hormiguero [4]. La optimización basada en Colonia de Hormigas (OCH) se ha convertido en un campo de investigación importante ya que se han desarrollado diversos modelos cada vez más sofisticados para solucionar de manera satisfactoria una gran variedad de problemas de optimización combinatorios.

### 4.3.1 Las colonias de hormigas naturales

Las hormigas son insectos sociales que viven en colonias y que, debido a su colaboración mutua, son capaces de mostrar comportamientos complejos y realizar tareas difíciles desde el punto de vista de una hormiga individual [4]. Lo más interesante del comportamiento de estos insectos, es la capacidad que tienen de encontrar la ruta más corta entre su hormiguero y los alimentos, ya que las hormigas son prácticamente ciegas.

La forma de lograr su cometido, es gracias a una sustancia química que van depositando en el camino llamada *feromona*. En el caso de que no se encuentren feromonas en el camino, las hormigas tienden a moverse de manera aleatoria. En el caso contrario, al encontrar las feromonas las hormigas tienden a seguir el rastro. Experimentos científicos demuestran que las hormigas prefieren de manera probabilística los caminos marcados con una concentración mayor de feromona [5].

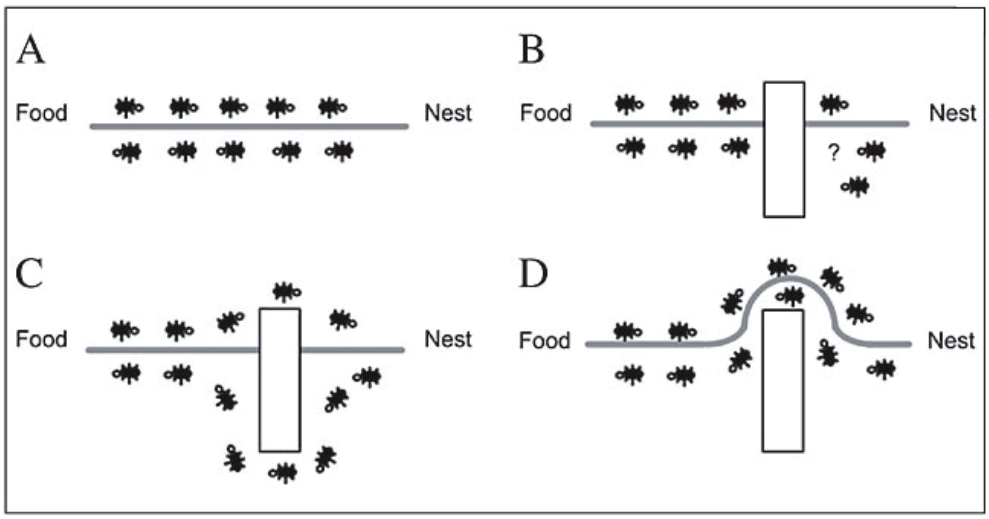


**Figura 4.2: Comportamiento de las hormigas al encontrar un obstáculo**

En la figura 4.2 se aprecia el comportamiento de las hormigas, primero siguen su camino hacia el alimento hasta que encuentran un obstáculo. Luego aleatoriamente elijen una ruta, pero al ser una de las dos más corta, más rápido es el retorno por lo que la carga de feromona es ligeramente mayor. Lo que incide en la decisión que tomaran las siguientes hormigas. Además, los rastros de feromona eventualmente comienzan a evaporarse, por lo que los caminos menos prometedores comenzaran a perder fuerza al ser menos visitados.

### 4.3.2 De las hormigas naturales a la metaheurística de OCH

Los algoritmos de OCH se basan en el comportamiento de las colonias de hormigas reales para resolver los problemas de optimización combinatorios, es decir, se basan en hormigas artificiales que se comunican mediante rastros de feromonas artificiales.

Estos algoritmos son principalmente *algoritmos constructivos*, ya que en cada iteración las hormigas van construyendo una solución recorriendo un *grafo de construcción*. Cada arista del grafo viene a representar un posible camino a seguir por la hormiga artificial y tiene asociado dos tipos de información que ayudan a guiar el movimiento hacia el siguiente nodo:

- **Información Heurística:** Mide la preferencia heurística de moverse desde un nodo r a un nodo s, se denota por $\eta_{rs}$. Esta información no puede ser modificada durante la ejecución. Sin embarco, en problemas dinámicos como el BACP, si puede cambiar.

- **Información de los rastros de feromona artificiales:** Imita a la feromona real que depositan las hormigas, se denota por $\tau_{rs}$. Esta información es modificada durante la ejecución del algoritmo.

### 4.3.3 La hormiga artificial

La hormiga artificial es un agente computacional simple que intenta construir soluciones posibles explotando los rastros de feromona disponibles y la información heurística. La hormiga artificial presenta las siguientes propiedades [6]:

- Busca soluciones validas de costo mínimo para el problema a solucionar.
- Tiene una memoria que almacena información sobre el camino seguido hasta el momento.
- Tiene un estado inicial $\delta_{inicial}$.
- Comienza en el estado inicial moviéndose hacia estados validos, construyendo la solución incrementalmente.
- Puede moverse a cualquier nodo de su vecindario posible.
- El movimiento se lleva a cabo aplicando una regla de transición que depende de los rastros de feromona, la información heurística y las restricciones del problema.
- Se puede actualizar el rastro de feromona $\tau_{rs}$ cuando se mueve desde el nodo r a un nodo s. Este proceso se llama *actualización en línea de los rastros de feromona paso a paso*.
- El procedimiento de construcción acaba cuando se satisface alguna condición de parada, normalmente cuando se alcanza un estado objetivo.
- Una vez construida la solución, se puede reconstruir el camino recorrido y actualizar los rastros de feromona de los arcos visitados utilizando un proceso llamado *actualización en línea a posteriori.*

### 4.3.4 Modo de funcionamiento y estructura genérica de un algoritmo de OCH

El modo de operación básico de un algoritmo de OCH es el siguiente: las m hormigas artificiales se mueven a través de estados adyacentes siguiendo una regla de transición que depende de la información heurística y los rastros de feromona. Al moverse por el grafo de construcción las hormigas van construyendo incrementalmente soluciones. Opcionalmente pueden depositar feromona al momento de atravesar un arco. Una vez que la hormiga construye una solución, ésta se evalúa y se deposita feromona dependiendo de la calidad de la solución.

La estructura genérica del algoritmo de Optimización por Colonia de Hormigas (OCH) contempla los siguientes procedimientos:

- **Inicialización de Parámetros**: Se fija el valor inicial de los rastros de feromona, que dependen del número de hormigas de la colonia y los pesos de la información heurística y memorística. Además el procedimiento se inicializa nuevamente si se presenta un estancamiento.

- **Programación de actividades**: Este proceso consta de 3 partes (i) Generación y puesta en funcionamiento de las hormigas artificiales, (ii) Evaporación de feromona y (iii) Acciones del demonio. *Las acciones del demonio* son acciones opcionales que no

tienen relación con las hormigas naturales, y que implementan tareas desde una perspectiva global que no pueden ser llevadas a cabo por las hormigas por su perspectiva local.

- **Condición de fin**: Puede ser una condición de tiempo fijo, o bien hasta encontrar una solución aceptable.

```
1 Procedimiento Metaheuristica_OCH()
2     Inicializacion_de_parametros
3     mientras (criterio_de_termino_no_satisfecho)
4           Programacion_de_actividades
5                 Generacion_de_Hormigas_y_actividad()
6                 Evaporacion_de_feromona()
7                 Acciones_del_demonio(){opcional}
8           fin Programacion_de_actividades
9     fin mientras
10 fin Procedimiento

1 Procedimiento Generacion_de_Hormigas_y_actividad()
2     repetir en paralelo desde k=1 hasta m {numero de hormigas}
3           Nueva_Hormiga(k)
4     fin repetir en paralelo
5 fin Procedimiento

1 Procedimiento Nueva_Hormiga(id_hormiga)
2     inicializa_hormiga(id_hormiga)
3     L = actualiza_memoria_hormiga()
4     mientras (estado_actual ≠ estado_objetivo)
5           P = calcular_probabilidad_transicion(A,L,Ω)
6           siguiente_estado = aplicar_politica_decision(P,Ω)
7           estado_actual = siguiente_estado
8           si(actualización_feromona_en_linea_paso_a_paso)
9                 depositar_feromona_en_arco_visitado()
10          fin_si
11          L = actualizar_estado_interno()
12    fin mientras
13    si(actualización_feromona_en_linea_a_posteriori)
14          para cada arco visitado
15                depositar_feromona_en_arco_visitado()
16          fin para
17    fin si
18    liberar_recursos_hormiga(id_hormiga)
19 fin Procedimiento
```

**Figura 4.3: Pseudocódigo de un algoritmo de OCH genérico**

### 4.3.5 Pasos para resolver un problema mediante OCH

Los pasos que se deben seguir para atacar problemas usando esta metaheurística se puede resumir en las siguientes tareas:

- Representar el problema a través de un grafo ponderado que será recorrido por las hormigas para construir sus soluciones.
- Definir de manera apropiada el significado de los rastros de feromona $\tau_{rs}$.

- Definir de manera apropiada la información heurística ($\eta_{rs}$) de cada decisión que deben tomar las hormigas mientras construyen la solución.
- En lo posible implementar una búsqueda local eficiente para el problema que se desea solucionar.
- Refinar los parámetros del algoritmo de OCH.

## 4.3.6 Modelos de Optimización Basada en Colonia de Hormigas

En la literatura se han propuesto diversos modelos que siguen la metaheurística OCH, entre esos modelos se encuentran:

- Sistema de Hormigas (SH o Ant System)
- Sistema de Colonia de Hormigas (SCH o Ant Colony System)
- Sistema de Hormigas Max-Min (SHMM o Max-Min Ant System)
- Sistema de Hormigas con ordenación (Rank-Based Ant System)
- Sistema de la Mejor-Peor Hormiga (SMPH o Best-Worst Ant System)

### 4.3.6.1 Sistema de Hormigas

Fue el primer algoritmo desarrollado por Dorigo, Maniezzo y Colorni en el año 1991 [13], y presentaba tres variaciones que se diferenciaban por la forma de actualizar los rastros de feromona y correspondían a SH-Densidad, SH-Cantidad y SH-Ciclo. En SH-Densidad la feromona se depositaba mientras se construían las soluciones, pero esta cantidad era constante, a diferencia del SH-Cantidad donde dependía de la deseabilidad heurística de la transición ($\eta_{rs}$). El SH-Ciclo, a diferencia de los anteriores, la deposición de feromona se realiza una vez que la solución se completa. De las tres, la SH-Ciclo fue la que obtuvo mejores resultados y se le continúo llamando SH.

La actualización de las feromonas se realiza una vez que todas las hormigas han construido una solución, luego esos rastros se reducen en un factor constante (evaporación). Luego cada hormiga de la colonia deposita una cantidad de feromona relacionada a la calidad de la solución.

Las soluciones son generadas de la siguiente forma: En cada paso de construcción, la hormiga k escoge ir al siguiente nodo con una probabilidad que se calcula con la siguiente fórmula:

$$p_{rs}^{k} = \frac{[\tau_{rs}]^{\alpha}[\eta_{rs}]^{\beta}}{\sum_{u \in N_r^K}[\tau_{ru}]^{\alpha}[\eta_{ru}]^{\beta}}, s \in N_k(r) \qquad (4)$$

En el caso que $s \notin N_k(r)$ la probabilidad se hace 0. Donde $N_k(r)$ es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo r, y $\alpha$, $\beta$ son dos parámetros que ponderan la importancia relativa de los rastros de feromona y la información heurística. Cada

hormiga almacena la secuencia que ha seguido y su memoria $L_k$ se utiliza para determinar $N_k(r)$ en cada paso de construcción.

Los parámetros α y β cumplen una función muy importante en el desarrollo del algoritmo, ya que si α=0, quiere decir que las hormigas solo se guían a través de la información heurística, haciéndolo similar a un algoritmo voraz[4] probabilístico clásico. Por otro lado, si β=0, se tomarían en cuenta solo los rastros de feromona en la elección de la hormiga, lo que llevaría rápidamente a que las hormigas tomen siempre los mismo caminos, construyendo la misma solución. Esto provoca un estancamiento en un óptimo local. Debido a esto, es preciso encontrar un equilibrio entre los valores de ambos parámetros.

La evaporación de la feromona en cada arco viene dado por el siguiente factor constante:

$$\tau_{rs} \leftarrow (1-\rho)\tau_{rs} \tag{5}$$

Donde ρ ϵ ]0,1[ es la tasa de evaporación. Luego cada hormiga debe recorrer nuevamente el camino que ha seguido para depositar una cantidad de feromona $\Delta\tau_{rs}^{k}$ en cada arco recorrido:

$$\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau_{rs}^{k}, \forall a_{rs} \in S_k \tag{6}$$

Donde $\Delta\tau_{rs}^{k} = f(C(S_k))$, es decir, la cantidad de feromona que se deposita depende de la calidad $C(S_k)$ de la solución $S_k$ construida por la hormiga k.

[4] Tipo de algoritmo que para resolver un problema sigue una metaheurística que consiste en elegir la opción optima en cada paso local con la esperanza de llegar a una solución general óptima.

```
1 Procedimiento Nueva_Hormiga(id_hormiga)
2     k = id_hormiga; r = generar_estado_inicial; Sk = r
3     Lk = r
4     mientras (estado_actual ≠ estado_objetivo)

5           para cada s ϵ Nk(r) hacer  p_rs^k = [τ_rs]^α [η_rs]^β / Σ_{u∈N_r^k} [τ_rs]^α [η_rs]^β

6           siguiente_est = aplicar_politica_de_decision(P, Nk(r))
7           r = siguiente_est; Sk=<Sk,r>
8           ---
9           Lk=Lk U r
10    fin mientras
11    Evaporacion_de_feromona()
12    para cada arco a_rs ϵ Sk
13          τrs= τrs + f(C(Sk))
14    fin para
15    liberar_recursos_hormiga(id_hormiga)
16 fin Procedimiento
```

**Figura 4.4: Procedimiento Nueva_hormiga para el Sistema de hormigas**

Los creadores del SH además crearon una forma mejorada llamada SH-Elitista donde además de depositar feromona en las aristas de las respectivas soluciones, el demonio realiza una deposición adicional de feromona en las aristas que pertenecen a la mejor solución encontrada hasta el momento.

### 4.3.6.2 El sistema de Colonia de Hormigas

El SCH fue una de las primeras variaciones del SH que introduce tres modificaciones importantes respecto del modelo anterior:

- Usa una regla de transición distinta, la que se denomino *regla proporcional pseudoaleatoria*. Sea k una hormiga situada en el nodo r, $q_0 \in [0,1]$ un parámetro y q un valor aleatorio entre [0,1], el siguiente nodo s se elije aleatoriamente mediante la siguiente distribución de probabilidad:

Si ($q \leq q_0$): $$p_{rs}^k = 1, \text{ si } s = \arg\max_{u \in N_k(r)} \{\tau_{rs}^\alpha \cdot \eta_{ru}^\beta\} \quad (7)$$

Si no ($q > q_0$): $$p_{rs}^k = \frac{[\tau_{rs}]^\alpha [\eta_{rs}]^\beta}{\sum_{u \in N_r^k} [\tau_{rs}]^\alpha [\eta_{rs}]^\beta}, s \in N_k(r) \quad (8)$$

Cuando $q \leq q_0$, explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona. Si $q > q_0$ se aplica una exploración controlada.

- Solo el demonio puede actualizar los rastros de feromona. El SCH solo considera una hormiga concreta, la que generó la mejor solución global ($S_{mejor\text{-}global}$).
  La evaporación solo se aplica a las conexiones de la solución:

$$\tau_{rs} \leftarrow (1-\rho)\tau_{rs}, \ \forall a_{rs} \in S_{mejor-global} \tag{9}$$

  El demonio deposita feromona usando la siguiente regla:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \cdot f(C(S_{mejor-global})), \ \forall a_{rs} \in S_{mejor-global} \tag{10}$$

- Las hormigas aplican una actualización en línea paso a paso de los rastros de feromona que favorece a la generación de soluciones distintas a las ya encontradas, se aplica la siguiente regla:

$$\tau_{rs} \leftarrow (1-\varphi) \cdot \tau_{rs} + \varphi \cdot \tau_0 \tag{11}$$

Donde φ ϵ (0,1] es un segundo parámetro de decremento de feromona. La regla de actualización en línea paso a paso incluye tanto evaporación como deposición de la misma.

```
1 Procedimiento Nueva_Hormiga(id_hormiga)
2     k = id_hormiga; r = Generar_Estado_Inicial; Sk = r; Lk = r;
3     mientras(estado_actual ≠ estado_objetivo)
4           para cada s ϵ Nk(r) calcular brs = τrs · ηrs^β
5           q = generar_valor_aleatorio_entre_0_1
            si(q ≤ q0) siguiente_estado = max(brs, sϵNk(r))
            si no para cada s ϵ Nk(r) hacer
                                  p_rs^k = b_rs / Σ_{u∈N_k(r)} b_ru
                  sig_est = aplicar_política_de_decisión(P, Nk(r))
            fin si
6           r = sig_est; Sk=<Sk, r>
7           τrs = (1−φ)·τrs + φ·τ0 ;  Lk = Lk ∪ r
8     fin mientras
9 fin Procedimiento
```

**Figura 4.5: Procedimiento Nueva_Hormiga para modelo SCH**

```
1 Procedimiento Acciones_del_demonio()
2     para cada S_k hacer búsqueda_local(S_k)
3     S_mejor-actual = mejor_solucion(S_k)
4     si(mejor(S_mejor-actual, S_mejor-global))
5           S_mejor-global=S_mejor-actual
6     fin si
7     para cada arista a_rs ∈ S_mejor-global hacer
8           {se ejecuta el procedimiento Evaporación_de_feromona()
            se lanza y evapora feromona en la arista a_rs:
            τ_rs = (1 − ρ) · τ_rs }
9           τ_rs = τ_rs + ρ · f(C(S_mejor−global))
10    fin para
11 fin Procedimiento
```

**Figura 4.6: Procedimiento Acciones_de_demonio en SCH**

### 4.3.6.3 El sistema de hormigas Max-Min

El SHMM es una de las extensiones del SH que muestra mejor rendimiento. Se extiende del SH en los siguientes aspectos:

- La actualización de feromona se hace fuera de línea, de manera similar al SCH. Después de que cada hormiga construye la solución cada rastro de feromona sufre una evaporación:

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs} \tag{12}$$

  Y luego se deposita feromona usando la siguiente fórmula:

$$\tau_{rs} \leftarrow \tau_{rs} + f(C(S_{mejor})), \forall a_{rs} \in S_{mejor} \tag{13}$$

  La mejor hormiga a la que se le permite añadir feromona puede ser la que tiene la mejor solución de la iteración o la mejor global.

- Los valores posibles para los rastros de feromonas se encuentran limitados a [$\tau_{min}$, $\tau_{max}$], por lo que la probabilidad de estancamiento es mínima al darle a cada enlace una probabilidad de ser escogida. Existen diversas heurísticas que permiten fijar los valores máximo y mínimo. A causa de la evaporación de la feromona, el nivel máximo que esta alcanza está limitado a $\tau^*_{max} = 1/(\rho \cdot C(S^*))$, donde S* es la solución optima. Entonces si se reemplaza $S_{mejor\text{-}global}$ por S* en la ecuación $\tau^*_{max}$ se puede estimar $\tau_{max}$. Para $\tau_{min}$ se puede escoger un valor constante menor que $\tau_{max}$.

  Para aumentar la exploración de nuevas soluciones, el SHMM re-inicializa los rastros de feromona.

- En vez de inicializar los rastros de feromonas a una cantidad pequeña, el SHMM los inicializa estimando el máximo permitido para un rastro. Esto lo puede hacer reemplazando S* en la fórmula del $\tau^*_{max}$ por un S' generado por una heurística voraz.

```
1 Procedimiento Acciones_del_demonio()
2     para cada Sk hacer búsqueda_local(Sk)
3     Smejor-actual = mejor_solucion(Sk)
4     si(mejor(Smejor-actual, Smejor-global))
5           Smejor-global=Smejor-actual
6     fin si
7     Smejor = decisión(Smejor-global, Smejor-actual)
8     para cada arista ars ∈ Smejor hacer
9           τrs = τrs + f(C(Smejor))
10          si(τrs < τmin)  τrs = τmin
11    fin para
12    si(condición_de_estancamiento)
13          para cada arista ars hacer τrs = τmax
14    fin si
15 fin Procedimiento
```

**Figura 4.7: Procedimiento Acciones_del_demonio en SHMM**

### 4.3.6.4 El Sistema de Hormigas con ordenación

En este modelo [4] se ordenan las hormigas al momento de actualizar los rastros de feromona, que el demonio realiza fuera de línea:

- Se ordenan de peor a mejor las m hormigas según la calidad de sus soluciones ($S'_1$,…$S'_m$), siendo $S'_1$ la mejor solución de la iteración.

- El demonio deposita feromona en las conexiones donde han pasado las σ-1 mejores hormigas (σ: hormigas elitistas). La cantidad de feromona depositada depende directamente del orden de la hormiga y la calidad de la solución.

- Las conexiones por las que ha pasado la mejor hormiga global reciben una cantidad adicional de feromona que depende únicamente de la calidad de dicha solución.

La regla que se aplica para la actualización de feromona cuando los rastros de feromona ya han sido evaporados en cada arista es la siguiente:

$$\tau_{rs} \leftarrow \tau_{rs} + \sigma \cdot \Delta\tau_{rs}^{mg} + \Delta\tau_{rs}^{orden} \tag{14}$$

Donde:

$$\Delta\tau_{rs}^{mg} = f(C(S_{mejor-global})), \forall a_{rs} \in S_{mejor-global} \tag{15}$$

$$\Delta\tau_{rs}^{orden} = \sum_{\mu=1}^{\sigma-1}(\sigma-\mu)\cdot f(C(S'_{\mu})), \forall a_{rs} \in S'_{\mu} \quad (16)$$

```
1 Procedimiento Acciones_del_demonio()
2     para cada Sk hacer Busqueda_local(Sk) {opcional}
3     ordenar (S1,…, Sm) en orden decreciente según
                  La calidad de la solución: (S'1,…, S'm)
4     si(mejor(S'1, Smejor-global))
5           Smejor-global = S'1
6     fin si
7     desde μ = 1 hasta (σ-1) hacer
8           para cada arista ars ∈ S'μ hacer
9                 τrs = τrs + (σ − μ)·f(C(S'μ))
10          fin para
11    fin desde
12    para cada arista ars ∈ Smejor-global hacer
13          τrs = τrs + σ·f(C(Smejor−global))
14    fin para
15 fin Procedimiento
```

**Figura 4.8: Procedimiento Acciones_del_demonio para SHordenacion**

### 4.3.6.5 El Sistema de la Mejor – Peor Hormiga

El SMPH es el último modelo que se analizó en este trabajo y es otra extensión del SH. El SMPH utiliza la misma regla de transición que el SH y la misma regla de evaporación que en el $SH_{ordenación}$ y el SHMM. Además considera la explotación sistemática de optimizadores locales para mejorar las soluciones de las hormigas, tal como lo hace el SHMM. Las acciones del demonio que realiza el SMPH son tres:

- *La regla mejor-peor de actualización de rastros de feromona*, refuerza las aristas que se encuentran en la mejor solución global y penaliza cada conexión que se encuentre en la peor solución generada hasta el momento ($S_{peor\text{-}actual}$) realizando una evaporación adicional, la regla de actualización sería la siguiente:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho\cdot f(C(S_{mejor-global})), \forall a_{rs} \in S_{mejor-global} \quad (17)$$

$$\tau_{rs} \leftarrow (1-\rho)\cdot\tau_{rs}, \forall a_{rs} \in S_{peor-actual} \text{ y } a_{rs} \notin S_{mejor-global} \quad (18)$$

- *Se realiza una mutación en los rastros de feromona* para introducir diversidad en el proceso de búsqueda. En el SMPH original se aplicaba un operador que alteraba los rastros de feromona añadiendo o restando la misma cantidad por cada iteración.

El rango de mutación (mut (it, $\tau_{umbral}$)), que depende de la medida de los rastros de feromona en las transiciones de la mejor solución global, $\tau_{umbral}$, es más suave en la primeras etapas donde no hay riesgo de estancamiento y luego mas fuerte donde hay más riesgo de estancamiento. La regla es la siguiente:

$$\tau_{rs} \leftarrow \tau_{rs} + mut(it, \tau_{umbral}), \text{ si a=0} \quad (19)$$

$$\tau_{rs} \leftarrow \tau_{rs} - mut(it, \tau_{umbral}), \text{ si a=1} \quad (20)$$

Donde *a* es un valor aleatorio en {0,1} e *it* es la iteración actual.

- Se considera la *reinicialización de los rastros de feromona* cuando se estanca la búsqueda. Los parámetros para decidir cuándo hacer la reinicialización han sido dos: En las primeras versiones se comprobaba el porcentaje de diferencia entre los arcos de la mejor solución y la peor solución era menor a un valor umbral. En versiones más recientes se comprueba el porcentaje de iteraciones del algoritmo sin haber obtenido una mejora en la mejor solución.

```
1 Procedimiento Acciones_del_Demonio()
2     para cada arco S_k hacer Busqueda_local(S_k)
3     S_mejor-actual = mejor_solucion(S_k)
4     si (mejor ( S_mejor-actual, S_mejor-global) )
5           S_mejor-global = S_mejor-actual
6     fin si
7     para cada arista a_rs ∈ S_mejor-global hacer
8           τ_rs = τ_rs + ρ · f(C(S_mejor-global)) ; suma = suma + τ_rs
9     fin para
10    τ_umbral = suma / |S_mejor-global|
11    S_peor-actual = peor_solucion(S_k)
12    para cada arista a_rs ∈ S_peor-global y a_rs ∉ S_mejor-global hacer
13          τ_rs = (1 − ρ) · τ_rs
14    fin para
15    mut = mut(it, τ_umbral)
16    para cada nodo / componente r ∈ {1,…,l}
17          z = valor_aleatorio_[0,1]
18          si (z ≤ P_m)
19                s = valor_aleatorio_[1,…,l]
20                a = valor_aleatorio_{0,1}{0 || 1}
21                si(a = 0) τ_rs = τ_rs + mut
22                si no τ_rs = τ_rs − mut
23          fin si
24    fin para
25    si (condición_de_estancamiento)
26          para cada arista a_rs hacer τ_rs = τ_0
27    fin si
28 fin Procedimiento
```

**Figura 4.9: Procedimiento Acciones_del_Demonio para SMPH**

## 4.4 Búsqueda local

La búsqueda local se basa en lo que es quizás el método más antiguo del área de optimización, la prueba y error. La idea es simple y natural, de hecho, es sorprendente lo exitoso que resulta ser la búsqueda local en una gran serie de problemas de optimización combinatorios [9].

Dada una instancia (F, c) donde "F" es cualquier conjunto dentro del dominio de puntos factibles y "c" es la función de costos, se elije un vecindario:

$N : F \rightarrow 2^F$

Que se busca en el punto $t \in F$ para mejoras usando la subrutina:

$$mejora(t) = \begin{cases} \text{Un s} \in N(t) \; donde \; \text{c(s)} < \text{c(t)}, & \text{si existe} \\ No, \text{ en caso contrario} \end{cases} \tag{21}$$

La idea es iniciar con una solución generada aleatoriamente hallada con algún otro algoritmo, para luego aplicar una transformación de algún conjunto dado de transformaciones para mejorar la solución actual. Luego la solución mejorada se convierte en la solución actual y este proceso se repite hasta que ninguna transformación del conjunto mejore la solución actual.

La solución encontrada no necesariamente será la óptima, ya que depende de las transformaciones aplicadas. Cuando no es posible obtener un mejoramiento en la solución el algoritmo se detiene en un óptimo local.

```
1 Procedimiento Busqueda_Local()
2     seleccionar s ϵ S
3     opt = s
4     Mientras (no termine) hacer
5           seleccionar s' vecindario de s
6           s = s' (mover a s')
7           Si (f(s)> f(opt)) hacer
8                 opt = s {en caso de maximizar}
9           fin si
10    fin mientras
11 fin procedimiento
```

**Figura 4.10: Algoritmo de búsqueda local genérico**

# 5 Propuesta de solución

Para resolver el problema del balanceo de mallas curriculares se utilizó el modelo de optimización de hormigas, SMPH. Ya que en los estudios realizados por [4] se demostró que este modelo en particular y bajo circunstancias similares, es el que logró mejores resultados, incluso cuando los valores de sus parámetros no eran los más adecuados, por lo que se considera un algoritmo robusto.

## 5.1 Esquema de solución propuesto

El esquema de solución propuesto, como se puede ver en la figura 5.1, recibe una instancia del BACP como entrada, que pasa por el algoritmo SMPH donde se hace la asignación de cursos a un periodo. Luego internamente se ejecuta una búsqueda local y otros procedimientos, como se explico en la sección 3.3.6.5, que generan la solución optimizada.

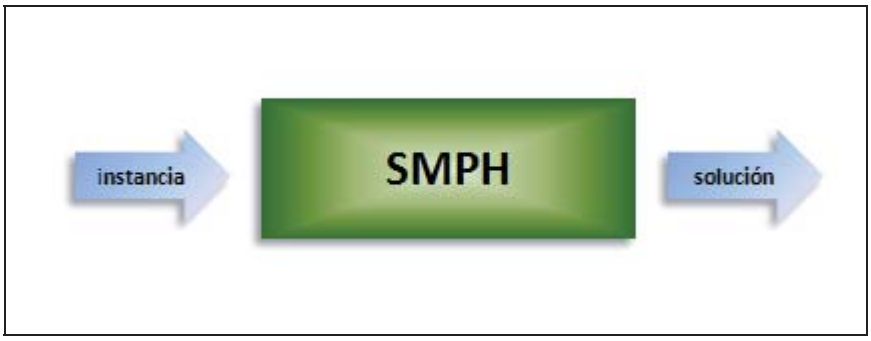


**Figura 5.1: Esquema de solución propuesto**

En otras palabras, es el SMPH el que se encarga, mediante el paso de cada hormiga por el grafo de construcción, de generar el emparejamiento inicial de cada curso con un periodo, y luego de construir soluciones que mediante búsquedas locales, evaporación y deposición de feromona, entregan una solución óptima al BACP.

Las principales funcionalidades que presenta el modelo para resolver el BACP son las siguientes:

- **Inicializar parámetros**: Es importante para la resolución del problema la correcta elección de los parámetros del algoritmo, ya que de estos depende la calidad de la solución que construyan las hormigas. Los parámetros del algoritmo son los siguientes:

    - Número de hormigas.
    - Número de iteraciones.
    - Tasa de evaporación ($\rho$).
    - Ponderador de feromona ($\alpha$).
    - Ponderador de la Información Heurística ($\beta$).

- Probabilidad de mutación ($P_m$)
- Fuerza de mutación ($\sigma$)

- **Inicializar matriz de feromona**: Al momento de comenzar a construir las soluciones se debe inicializar los rastros de feromona, para esto se genera una solución inicial que solo se preocupa de respetar los prerrequisitos de cada curso. Además, si la mejor solución encontrada se estanca, se puede reinicializar los rastros de feromona con el fin de buscar nuevos caminos y no mantenerse en un óptimo local. Permitiendo así una mayor exploración del espacio de búsqueda de las soluciones.

- **Crear colonia de hormigas**: Comprende la creación de cada hormiga perteneciente a la colonia y posterior asignación de un periodo para cada asignatura siguiendo la regla de transición basada en una regla de probabilidad (ecuación 20).

- **Realizar búsqueda local**: Cada hormiga implementará una búsqueda local, la cual mejorará la calidad de las soluciones en el caso que se encuentre. Para esto se realizan movimientos de vecindarios los cuales se explican en la sección 5.5.

- **Actualizar matriz de feromona**: Una vez que se cuenta con una solución optimizada el siguiente paso es la actualización de la matriz de feromona. Esta se da de cuatro maneras:

    - Evaporación: Se realiza en toda la matriz de feromona usando la tasa de evaporación (ecuación 4).
    - Deposición: Cada hormiga deposita una cantidad de feromona asociada a la calidad de la solución que generó (ecuación 5).
    - Deposición de la mejor hormiga: Se selecciona a la hormiga que género la mejor solución global en cada iteración y para cada nodo donde esta pasó se deposita una cierta cantidad de feromona (ecuación 16).
    - Evaporación de la peor hormiga: Se selección a la hormiga que género la peor solución por iteración y por cada nodo perteneciente al camino generado y que a su vez no pertenezca a la mejor solución global, se evapora el rastro de feromona (ecuación 17).

- **Realizar mutación de feromona**: La idea de mutación proviene de los algoritmos evolutivos y se usa con el fin de dar diversidad a las soluciones, de esta manera se puede evitar estancamiento en óptimos locales, permitiendo así explorar mejor el espacio de búsqueda de las soluciones.

A continuación se muestra un diagrama de flujo de la secuencia de pasos que realiza el Sistema de la Mejor-Peor Hormiga para resolver el problema de balanceo de mallas curriculares.

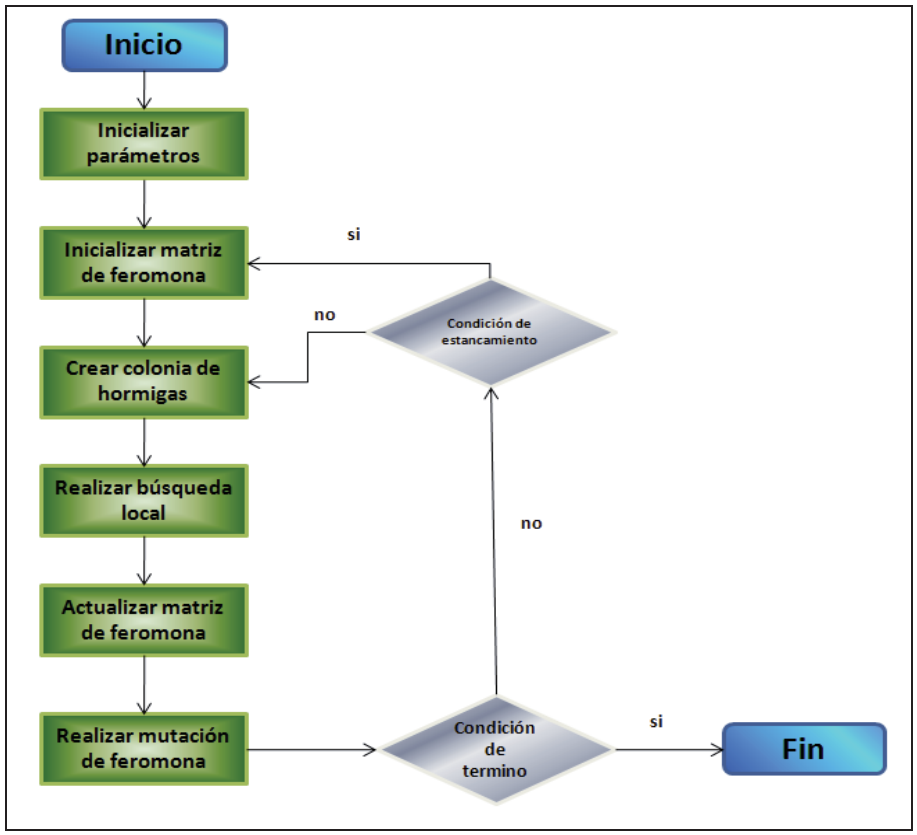


**Figura 5.2: Diagrama de flujo del funcionamiento del algoritmo**

## 5.2 Descripción del problema

El BACP es un problema de asignación de C cursos a un periodo T, el cual puede ser de 8, 10 o 12 periodos dependiendo del plan de estudio que se quiera resolver. Las restricciones que posee pueden clasificarse en duras o blandas. Las *restricciones duras* que presenta el problema son las siguientes:

- Ningún alumno puede cursar una asignatura sin haber aprobado su pre-requisito con anterioridad.
- Todos los cursos deben ser asignados a un periodo.
- El número máximo de periodos debe ser el indicado para cada instancia del problema, por ejemplo, el bacp8 no puede tener más de 8 periodos académicos

Estas restricciones duras siempre deben cumplirse para que la solución generada pueda ser válida. Luego de satisfacer la restricción dura se deberá optimizar la solución satisfaciendo las restricciones blandas, que para este problema son las siguientes:

- El número de cursos asignado a un periodo determinado debe ser mayor que el mínimo establecido.

- El número de cursos asignado a un periodo determinado debe ser menor que el máximo establecido.
- El número de créditos asignado a un periodo determinado debe ser mayor que el mínimo establecido.
- El número de créditos asignado a un periodo determinado debe ser menor que el máximo establecido.

Se busca minimizar la cantidad de restricciones blandas que se violen. Ya que difícilmente se logre una solución que satisfaga todas las restricciones blandas.

## 5.3 Estructura de resolución

Como se dijo anteriormente, el BACP es un problema de asignación y la meta heurística OCH es eficiente resolviendo problemas del tipo problema del camino óptimo. Por lo que se transformara el tipo de problema, para esto se debe crear un grafo de construcción que las hormigas deben seguir para crear sus soluciones. En la figura 5.2 se puede ver un grafo de construcción donde $C_1$ hasta $C_c$ representa cada uno de los cursos y $P_1$ hasta $P_p$ cada uno de los periodos. Las hormigas deberán construir sus soluciones asociando un curso a un periodo.

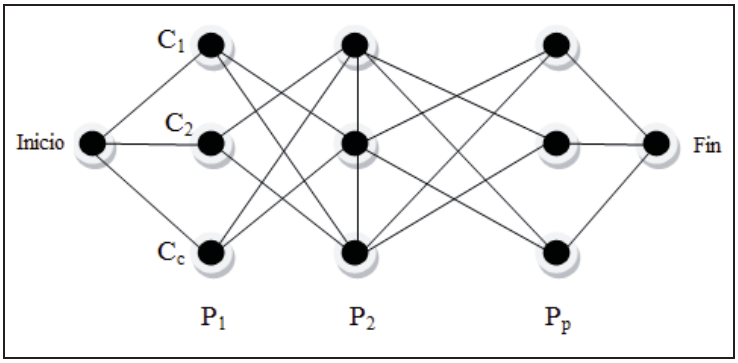


**Figura 5.3: Grafo de construcción**

Para implementar la solución se evaluaron dos representaciones alternativas para el grafo de construcción: la primera opción es una matriz binaria de CxP (cursos por periodos), donde cada casilla tendrá el valor 0, en el caso de que el curso i no tenga asignado un periodo j, o el valor 1 en el caso de que el curso i tenga asignado un periodo j. En la figura 5.4 se ilustra la representación matricial del grafo donde el primer curso es asignado al segundo periodo, el segundo curso al primer periodo, y así sucesivamente.

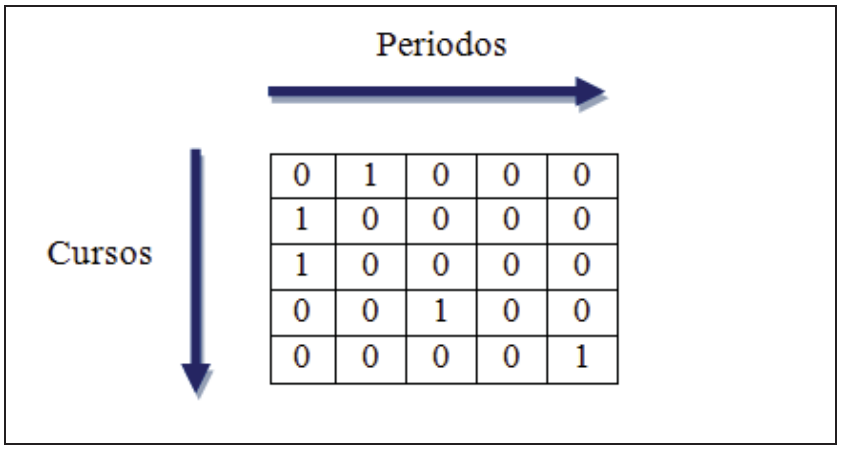


**Figura 5.4: Representación matricial del grafo de construcción**

La segunda opción de representación es un arreglo de enteros de tamaño C (número de cursos), donde cada espacio del vector representa un curso, y su contenido es el periodo que se le asigno. Por ejemplo, siguiendo la figura 5.5 se puede ver como al curso 1 se le asigna el primer periodo, al curso 2 el 1er periodo, al curso 3 el 2do periodo y así hasta llegar a asignar un periodo al último curso.

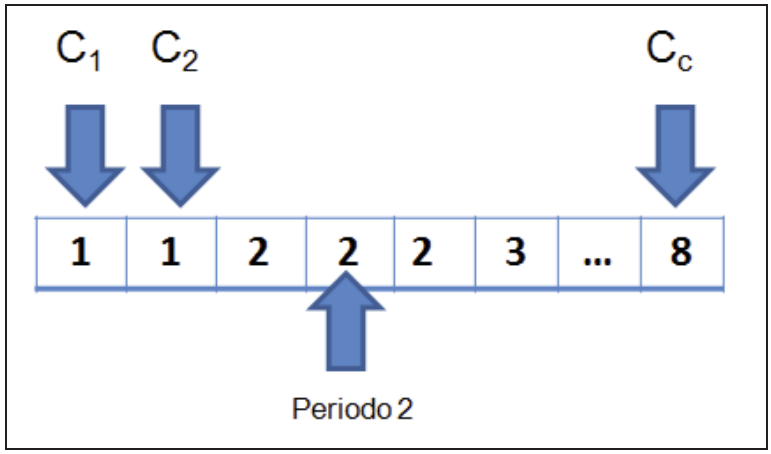


**Figura 5.5: Representación vectorial del grafo de construcción**

Para caminar a través del grafo de construcción las hormigas deben seguir una regla de transición que viene dada por una probabilidad, la cual depende de la preferencia heurística y del rastro de feromona asociado. La regla de transición es la siguiente:

$$P(c, p_i) = \frac{[\tau_{(c,p_i)}]^{\alpha} \cdot [\eta_{(c,p_i)}]^{\beta}}{\sum_{u \in P} [\tau_{(c,p_u)}]^{\alpha} \cdot [\eta_{(c,p_u)}]^{\beta}} \quad (22)$$

Luego de tener definido el grafo y la regla de transición, se necesita definir la matriz de feromona y la información heurística. La matriz de feromona en vez de representar los rastros de feromona de un arco entre dos nodos, representa la carga de feromona que asociada a un nodo. Por lo que la matriz de feromona se verá como en la figura 5.6 donde $\tau_{11}$ corresponde a la carga de feromona correspondiente al curso 1 asignado al periodo 1, y así sucesivamente.

$$[\tau_{ij}] = \begin{bmatrix} \tau_{11} & \tau_{12} & \tau_{13} & \dots & \tau_{1p} \\ \tau_{21} & \tau_{22} & \tau_{23} & \dots & \tau_{2p} \\ \tau_{31} & \tau_{32} & \tau_{33} & \dots & \tau_{3p} \\ \dots & \dots & \dots & \dots & \dots \\ \tau_{c1} & \tau_{c2} & \tau_{c3} & \dots & \tau_{cp} \end{bmatrix}$$

**Figura 5.6: Matriz de feromona**

Para representar la información heurística se va a tomar una suma de las nuevas restricciones blandas que han sido violadas si se efectúa una asignación. La información heurística se decidirá por la siguiente regla:

$$\eta_{ij} = \frac{1}{1 + v_d(i, j) - v_a(i, j)} \tag{23}$$

Donde $v_d(i, j)$ es una función que indica la suma de las restricciones blandas que han sido violadas si es que un curso i se le asigna a un periodo j y $v_a(i, j)$ es una función que indica la suma de las restricciones blandas que han sido violadas *antes* de asignar un curso i a un periodo j.

$$v_d(i, j) = Ra_{ij} + Rc_{ij} \tag{24}$$

$$v_a(i, j) = Ra_{ij} + Rc_{ij} \tag{25}$$

$$Ra_{ij} = \begin{cases} 1, \text{si el número de cursos} \notin [\delta, \varepsilon] \\ 0, \text{si el número de cursos} \in [\delta, \varepsilon] \end{cases} \tag{26}$$

$$Rc_{ij} = \begin{cases} 1, \text{si el número de creditos} \notin [\beta, \gamma] \\ 0, \text{si el número de creditos} \in [\beta, \gamma] \end{cases} \tag{27}$$

Donde δ es el mínimo de cursos permitido por periodo, ε el máximo de cursos permitido por periodo, β el número mínimo de créditos permitido por periodo y γ el número máximo de créditos. Por lo tanto $Ra_{ij}$ evalúa si existe violación en la cantidad de asignaturas asignados a un periodo determinado y $Rc_{ij}$ mide la existencia de violación en la cantidad de

créditos que son asignados a un periodo académico. En la medida que mayor sea el valor de $v_d(i, j)$, menor será la preferencia heurística, por lo que habrá menos probabilidades de asignar el curso i al periodo j.

## 5.4 Descripción del algoritmo

A continuación se presenta la estructura general del algoritmo SMPH para la resolución al problema del BACP.

```
procedimiento smph_bacp()
      inicializar_matriz_feromona()
      mientras(condicion_de_termino_no_satisfecha)
            hormigas = crear_colonia_de_hormigas()
            evaporacion()
            mejor_actual = mejor_solucion(hormigas)
            busqueda_local(mejor_actual)
            si(mejor(mejor_actual,mejor_global))
                  mejor_global = mejor_actual
            fin_si
            tUmbral = calcular_tUmbral()
            peor_actual = peor_solucion(hormigas)
            evaporar_peor_solucion()
            mutacion(tUmbral)
            si(condicion_estancamiento)
                  reinicializar_matriz_feromona()
            fin_si
      fin_mientras
fin_procedimiento
```

**Figura 5.7: Estructura general del algoritmo SMPH aplicado a BACP**

Lo primero es ***inicializar la matriz de feromona*** con el valor $\tau_0 = 1/C(SolInicial)$, donde $C(SolInicial)$ es la calidad de una solución inicial, la cual se genera satisfaciendo únicamente la restricción dura de prerrequisitos. Luego se calcula la calidad de esta solución mediante la función C la cual se describe más adelante en esta sección, por lo que cada nodo quedara cargado con una cantidad $\tau_0$ de feromona que depende de la calidad inicial de solución.

```
procedimiento inicializar_matriz_feromona()
     desde i=1 hasta C hacer
          desde j=1 hasta P hacer
               matriz_feromona[i][j]=calidad_inicial()
          fin_desde
     fin_desde
fin_procedimiento
```

**Figura 5.8: Procedimiento inicializar_matriz_de_feromona**

Una vez inicializada la matriz de feromona se ingresa a un bucle que durará mientras se cumpla una cantidad de iteraciones determinada antes de iniciar la ejecución del algoritmo. Lo siguiente es ***crear una colonia de hormigas***, donde el número de hormigas también es determinado antes de que comience la ejecución del algoritmo. Esta función le permitirá además parear una asignatura con algún periodo permitido, es decir, con algún periodo de manera tal que se cumpla la relación de precedencia. A continuación el algoritmo que crea la colonia de hormigas.

```
funcion crear_colonia_de_hormigas()
     hormigas = nuevo_vector()
     desde i=1 hasta k hacer
          hormiga = nueva_hormiga()
          desde j=1 hasta C hacer
               periodo = regla_de_transicion(curso[j])
               hormiga.asignar_curso_periodo(curso[j],periodo)
          fin_desde
          hormiga.calidad_solucion = asignar_calidad_solucion()
          depositar_feromona(hormiga.calidad_solucion)
          hormigas[i]=hormiga
     fin_desde
     devolver hormigas
fin_funcion
```

**Figura 5.9: Función crear_colonia_de_hormigas**

Primero se crea un arreglo donde se almacenaran las distintas hormigas, cada hormiga tiene como datos la representación del grafo[5] y la calidad de su solución. La calidad de la solución viene dada por la mayor carga académica de todos los periodos académicos más un coeficiente de violación de las restricciones que no se hayan satisfecho. Luego, se inicia un bucle de k iteraciones (k hormigas que pertenecerán a la colonia) donde se crea una hormiga, y por cada curso se elije un periodo mediante la regla de transición para luego ser asignado a esa hormiga. Se evalúa la calidad de la solución para depositar los rastros de feromona de acuerdo a ese valor. La calidad viene dada por 2 componentes. El primero es la carga académica mayor asignada entre todos los periodos, y el segundo corresponde al costo, que corresponde al castigo generado por violación de restricciones, donde se multiplica por el valor 8 para darle un peso mayor. Mientras más grande es el 2do componente, peor será la calidad de la

[5] Esta puede ser la representación matricial (figura 5.4) o la vectorial (figura 5.5).

solución. Finalmente se guarda cada hormiga en el arreglo de hormigas y se retorna al algoritmo principal.

$$C(sol) = \frac{1}{Max\{c_1, c_2, ..., c_P\} + 8 \cdot \sum_{i=1}^{C} \sum_{j=1}^{P} costo(i, j)} \quad (28)$$

La regla de transición es una probabilidad (ecuación 22) que toma la decisión por la hormiga de qué camino seguir dentro del grafo de construcción. Se debe aplicar la regla a todos los nodos candidatos posibles, es decir, para un curso determinado se evalúan solo aquellos que respeten su regla de precedencia. El que obtenga una probabilidad mayor no determina que sea seleccionado ya que es solo una probabilidad, por lo que la elección se basa en una ruleta, donde cada probabilidad es almacenada y luego ejecutando un operador de ruleta se selecciona un nodo. Esto quiere decir que el par ($<C_i, P_j>$) con mayor probabilidad tiene más oportunidades de ser seleccionado pero no lo garantiza.

```
funcion regla_de_transicion(curso)
      probabilidad = 0
      valor = 0
      ruleta = crearRuleta()
      periodo_curso = buscar_periodo(curso)
      desde i = periodo_curso + 1 hasta numero_periodos_maximo hacer
            probabilidad += calcular_probabilidad()
            ruleta.añadir(probabilidad)
      fin_desde
      valor = aleatorio(0,...,probabilidad)
      desde j=1 hasta ruleta.tamaño() hacer
            si(valor <= ruleta[j])
                  retornar (j + periodo_curso +1)
            fin_si
      fin_desde
      retornar (periodo_curso + 1)
fin_funcion
```

**Figura 5.10: Función regla_de_transicion**

El segundo paso es realizar la ***búsqueda local*** a la mejor hormiga de la iteración. El procedimiento consiste en seleccionar los periodos que tenga la mayor y menor carga académica. Luego se crea un vector donde se almacenarán los distintos cursos asignados al periodo con más créditos. Esto se realiza con el fin de analizar las posibilidades de asignar uno de esos cursos al periodo que tengo menos créditos sin violar ninguna restricción. Finalmente se comprueba que esta nueva solución sea mejor que la anterior y en el caso de que lo sea, se asigna al mejor actual.

```
procedimiento busqueda_local(mejor_actual)
      hormiga = mejor_actual
      periodo_mayor = seleccionar_periodo_mas_creditos(mejor_actual)
      periodo_menor = seleccionar_periodo_menos_creditos(mejor_actual)
      cursos = vector()
      cursos = buscar_cursos_del_periodo(periodo_mayor)
      curso = seleccionar_curso(cursos)
      hormiga.asignar_curso_periodo(curso, periodo_menor)
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga, mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.11: Procedimiento búsqueda local**

El siguiente paso es la ***evaporación***, con el fin de explorar mejor el espacio de búsqueda y no estancarse en óptimos locales. El pseudocódigo es el siguiente donde se usa la ecuación 4 en cada posición de la matriz de feromona.

```
procedimiento evaporacion()
      desde i=1 hasta C hacer
         desde j=1 hasta P hacer
               matriz_feromona[i][j] = matriz_feromona[i][j]*(1-tasa_evaporacion)
         fin_desde
      fin_desde
fin procedimiento
```

**Figura 5.12: Procedimiento evaporación**

En cada iteración se selecciona a la hormiga que entregue la mejor solución de la colonia (figura 5.13) y esta es comparada con la mejor global, es decir, con la que ha entregado la mejor solución desde que comenzó la ejecución del algoritmo. En el caso que la calidad de la solución de la mejor hormiga actual supere la calidad de la solución de la mejor global, esta es reemplazada.

```
funcion mejor_solucion(hormigas[])
      calidad_solucion = 0, mejor = 0
      desde i=1 hasta k hacer
            si(hormigas[i].calidad_solucion > calidad_solucion)
                  calidad_solucion = hormigas[i].calidad_solucion
                  mejor = i
            fin_si
      fin_desde
      devolver hormigas[mejor]
fin_funcion
```

**Figura 5.13: Función mejor solución**

Lo siguiente es calcular el valor de $\tau_{umbral}$ que más tarde permitirá ejecutar la mutación en los rastros de feromona y además la actualización por depósito de ésta. El depósito se realiza usando la ecuación 17 que como se menciono anteriormente, realiza el depósito en función de la calidad de la mejor solución global. Luego los rastros de feromona se acumulan y se retornan a la función *calcular_tUmbral* donde se divide ese valor por la cantidad de nodos pertenecientes a la solución, que para este problema en particular siempre será el número de cursos[6].

```
funcion calcular_tUmbral()
      devolver depositar_mejor_solucion()/C
fin_funcion

funcion depositar_mejor_solucion()
      desde i=1 hasta C hacer
            desde j=1 hasta P hacer
                  si((curso i, periodo j) pertenece mejor_global)
                        matriz_feromona[i][j]+= tasa_evaporacion *
                        mejor_global.calidad_solucion
                        suma += matriz_feromona[i][j]
                  fin_si
            fin_desde
      fin_desde
      devolver suma
fin_funcion
```

**Figura 5.14: Funciones calcular_tUmbral y depositar_mejor_solucion**

La ***regla de la mejor-peor actualización de los rastros de feromona*** consiste en la deposición y la evaporación de los rastros de feromona, el depósito se aplica a la mejor solución global generada. Por otro lado, para la evaporación se debe utilizar todos los nodos por donde transitan las hormigas al generar la peor solución siempre y cuando no pertenezcan a la mejor global. La regla se aprecia en la ecuación 18 y los algoritmos se aprecian en la figura 5.15 y 5.16. Se elije la hormiga con la peor calidad de solución entre toda la colonia y luego se procede a evaporar en los puntos por donde no han transitado las hormigas al generar la mejor solución global.

[6] Ya que cada a cada curso debe asignársele un periodo académico.

```
funcion peor_solucion(hormigas[])
      calidad_solucion = 100, peor = 0
      desde i=1 hasta k hacer
            si(hormigas[i].calidad_solucion < calidad_solucion)
                  calidad_solucion = hormigas[i].calidad_solucion
                  peor = i
            fin_si
      fin_desde
      devolver hormigas[peor]
fin_funcion
```

**Figura 5.15: Función peor solución**

```
procedimiento evaporar_peor_solucion()
      desde i=1 hasta C hacer
            desde j=1 hasta P hacer
                  si((curso i, periodo j) pertenece peor_actual y
(curso i, periodo j) no pertenece a mejor_global)
                        matriz_feromona[i][j] *= (1-tasa_evaporacion)
                  fin_si
            fin_desde
      fin_desde
fin_procedimiento
```

**Figura 5.16: Procedimiento evaporar peor solución**

En la sección 4.3.6.5 se menciona que el SMPH posee tres características adicionales con respecto a otros modelos de optimización basados en hormigas: (I) La regla de la mejor-peor actualización de los rastros de feromona (II) La mutación y (III) Reinicialización de los rastros de feromona. Por el momento se describió la (I) y se hablo de la mutación y el cálculo de $\tau_{umbral}$. Lo que lleva a la descripción de la ***mutación***, que viene de la computación evolutiva con el fin de dar diversidad a las soluciones permitiendo explorar mejor el espacio de búsqueda.

```
procedimiento mutación()
      mut = calcular_coeficiente_mutacion(tUmbral)
      desde i=1 hasta C hacer
            si(aleatorio(0,...,1) < Pm )
                  j = aleatorio(1,P)
                  a = aleatorio(0,...,1)
                  si(a == 0)
                        matriz_feromona[i][j] += mut
                  sino
                        matriz_feromona[i][j] -= mut
                  fin_si
            fin_si
      fin_desde
fin_procedimiento
```

**Figura 5.17: Procedimiento mutación**

Primero se calcula el coeficiente de mutación (ecuación 29) el cual se sumará o restará dependiendo de un valor aleatorio $a \in [0,1]$. Para cada curso se elije un valor aleatorio que debe ser menor al parámetro $P_m$. Si se cumple esta condición se elije un periodo aleatoriamente ( $j \in [1,P]$), donde P es el número máximo de periodos permitidos para una cierta malla curricular, y luego se aplica la mutación correspondiente al par <curso, periodo>.

$$mut(it, \tau_{umbral}) = \frac{it - it_r}{Nit - it_r} \cdot \sigma \cdot \tau_{umbral} \tag{29}$$

Donde *it* es la iteración actual, $it_r$ la ultima iteración donde se realizo una reinicialización de los rastros de feromona, *Nit* la cantidad máxima de iteraciones y $\sigma$ que especifica el poder de la mutación con respecto al número de iteraciones desarrolladas hasta cierto momento. El operador de mutación es pequeño en la medida que se encuentran nuevas soluciones globales y comienza a aumentar hasta que se realiza la reinicialización. Con el correr de las iteraciones el numerador comienza a crecer y el denominador a disminuir ya que $it_r$ es cada vez mayor, a menos que no existan re inicializaciones. Esto permite explorar el espacio de búsqueda en un principio y sobre el final explotar.

Y finalmente la tercera característica incorporada en este algoritmo es la ***reinicialización de los rastros de feromona*** en el caso de que se estanque la solución en un óptimo local y viene dado por $Nit \cdot 0.2$, es decir, si la solución encontrada no mejora transcurrido el 20% del número total de iteraciones se aplicará el procedimiento de reinicialización de los rastros de feromona. La idea es que la matriz de feromona vuelva al estado en que se encontraba cuando se alcanzo la mejor global. De esta forma, se puede comenzar la búsqueda en un vecindario cercano a un posible óptimo en lugar de volver al vecindario inicial.

## 5.5 Movimientos de vecindario

Al momento de hacer la búsqueda local, con el fin de mejorar las soluciones, se deben hacer movimientos de vecindario y evaluar si la nueva solución tiene mejor calidad que la generada por la hormiga artificial. Se definen 6 movimientos de vecindario basados en los utilizados para el problema del *University Course Timetabling Problem (UCTP)* resuelto por [14]:

- Del 10% de los cursos (elegidos aleatoriamente) elegir el que viola más restricciones blandas y moverlo a un periodo que viole la menor cantidad de restricciones suaves.
- Elegir dos cursos aleatoriamente e intercambiar periodos.
- Elegir un curso al azar y moverlo a otro periodo elegido aleatoriamente.
- Seleccionar dos periodos al azar y mover todos los cursos de un periodo al otro.
- Mover un periodo eligiendo 2 periodos i y j al azar (i < j), luego mover todos los cursos del periodo i al periodo j, los cursos del periodo j al j-1 hasta que los del i+1 se muevan al i.
- Elegir un curso perteneciente al periodo que tenga más créditos y moverlo al periodo que tenga menos créditos, siempre cuando no viole ninguna restricción dura o blanda.

### 5.5.1 Movimiento de vecindario tipo 1

Para el primer movimiento de vecindario se toma el 10% de la cantidad total de cursos pertenecientes al plan académico y se selecciona aquel que viola la mayor cantidad de restricciones blandas (en caso de empate se elije el primer curso). Luego, se asigna esa asignatura al periodo donde genera el número menor de violaciones a las restricciones blandas del problema (en caso de empate se elije el periodo menor).

```
procedimiento movimiento_uno(mejor_actual)
      hormiga = mejor_actual
      cursos = vector()
      cursos = seleccionar_cursos(hormiga)
      curso = seleccionar_curso_mas_violaciones_blandas(hormiga,cursos)
      periodo = seleccionar_periodo_menos_violaciones_blandas(hormiga,curso)
      hormiga.asignar_curso_periodo(curso,periodo)
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga,mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.18: Procedimiento de búsqueda local utilizando el primer tipo de movimiento de vecindario**

En la figura 5.18 se muestra el algoritmo que realiza el primer movimiento de vecindario. Primero se crea una copia de la mejor solución actual de cada iteración. Esto se realiza con el fin de poder comparar posteriormente si la solución generada por el movimiento de vecindario logra superar la solución inicial en términos de calidad. Lo siguiente es seleccionar de manera aleatoria el 10% de los cursos para luego tomar aquel que viole la menor cantidad de restricciones blandas. Utilizando aquel curso, se elije aquel periodo donde

se viole la menor cantidad de restricciones en el caso de que sea asignado a ese semestre. Finalmente se determina la nueva calidad de solución generada. En caso de no haber mejoras se mantiene la solución inicial.

### 5.5.2 Movimiento de vecindario tipo 2

El segundo tipo de movimiento es el más simple. Toma dos cursos escogidos de manera aleatoria para luego intercambiar periodos.

```
procedimiento movimiento_dos(mejor_actual)
      hormiga = mejor_actual
      curso_uno = aleatorio(1,C)
      periodo_uno = obtener_periodo_del_curso(curso_uno)
      curso_dos = aleatorio(1,C)
      periodo_dos = obtener_periodo_del_curso(curso_dos)
      hormiga.asignar_curso_periodo(curso_uno,periodo_dos)
      hormiga.asignar_curso_periodo(curso_dos,periodo_uno)
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga,mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.19: Procedimiento de búsqueda local utilizando el segundo tipo de movimiento de vecindario**

De igual forma que el movimiento de vecindario tipo 1, se crea una copia de la mejor solución de la iteración para luego comparar la nueva solución con la actual y ser actualizado en el caso que corresponda.

### 5.5.3 Movimiento de vecindario tipo 3

El tercer movimiento de vecindario es similar al anterior. Se elige un curso al azar y se le asigna un periodo escogido aleatoriamente.

```
procedimiento movimiento_tres(mejor_actual)
      hormiga = mejor_actual
      curso = aleatorio(1,C)
      periodo = aleatorio(1,P)
      hormiga.asignar_curso_periodo(curso,periodo)
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga,mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.20: Procedimiento de búsqueda local utilizando el tercer tipo de movimiento de vecindario**

### 5.5.4 Movimiento de vecindario tipo 4

Consiste en seleccionar dos periodos de manera aleatoria y mover todos los cursos de un semestre a otro. El procedimiento se muestra en la figura 5.21:

```
procedimiento movimiento_cuatro(mejor_actual)
      hormiga = mejor_actual
      periodo_uno = aleatorio(1,P)
      periodo_dos = aleatorio(1,P)
      cursos_uno = vector()
      cursos_uno = buscar_cursos_periodo(periodo_uno)
      cursos_dos = vector()
      cursos_dos = buscar_cursos_periodo(periodo_dos)
      desde i=1 hasta cursos_uno.tamaño hacer
            hormiga.asignar_curso_periodo(cursos_uno[i],periodo_dos)
      fin_desde
      desde i=1 hasta cursos_dos.tamaño hacer
            hormiga.asignar_curso_periodo(cursos_dos[i],periodo_uno)
      fin_desde
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga,mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.21: Procedimiento de búsqueda local utilizando el cuarto tipo de movimiento de vecindario**

### 5.5.5 Movimiento de vecindario tipo 5

El quinto tipo de movimiento de vecindario consiste en escoger un periodo i y un periodo j de manera aleatoria (i < j) y se toma cada curso del periodo i para moverlo al periodo j. Luego se realiza la misma acción para cada curso del periodo j moviéndolos hacia el periodo j-1, los del j-1 hacia el j-2 hasta que los del i+1 se mueven al periodo i. El procedimiento que realiza el movimiento tipo 5 se puede apreciar en la figura 5.22 donde *intercambiar_periodos* es el proceso que se encarga de mover los periodos de la forma que se explicó anteriormente.

```
procedimiento movimiento_cinco(mejor_Actual)
      hormiga = mejor_actual
      periodo_uno = aleatorio(1,P)
      periodo_dos = aleatorio(1,P)
      intercambiar_periodos(hormiga, periodo_uno, periodo_dos)
      hormiga.calidad_solucion = asignar_calidad_solucion()
      si(mejor(hormiga,mejor_actual))
            mejor_actual = hormiga
      fin_si
fin_procedimiento
```

**Figura 5.22: Procedimiento de búsqueda local utilizando el quinto tipo de movimiento de vecindario**

Sin embargo, a diferencia del UCTP, el BACP tiene la regla de precedencia como restricción dura, lo que significa que al cambiar alguna asignación puede significar una violación de esta. Por lo que el movimiento de vecindario utilizado es el tipo 6 que es explicado en la sección 5.4 donde se describe paso a paso el algoritmo utilizado para la resolución del problema. En los movimientos tipo 3,4 y 5 la probabilidad de violar algún prerrequisito es alta, por lo que no se logra mejorar la calidad de las soluciones generadas por las hormigas.

## 5.6 Lenguaje de programación y entorno de trabajo

El prototipo fue desarrollado en el lenguaje de programación Java en su 2da versión, esto debido a su simpleza, ya que elimina la complejidad de lenguajes como C, específicamente en el manejo de memoria dinámica, por lo que la preocupación se limita a la resolución del problema y no a su programación. Además permite trabajar con objetos lo que fomenta la reutilización de código, relaciona el sistema al mundo real y agiliza el desarrollo del prototipo.

Se utilizó para desarrollar el prototipo el IDE (entorno de desarrollo integrado) Netbeans IDE 7.0.1 (JDK 1.6) con el fin de agilizar el proceso de desarrollar código y su interfaz gráfica. Además se utilizó como procesador un Intel Core 2 Duo T8300 de 2.40 GHz para la ejecución de las pruebas.

# 6 Prototipo y análisis de resultados

En esta sección se muestra el prototipo que resuelve el BACP, su interfaz y funcionalidades. Luego se analizaron los resultados obtenidos en la resolución de las instancias benchmark de la CSPLib (8, 10 y12 periodos) y además algunas instancias reales. Entre ellas se encuentran las mallas de las carreras de Ingeniería Ejecución Informática e Ingeniería Civil Informática de la Pontificia Universidad Católica de Valparaíso e Ingeniería Informática de la Universidad de Playa Ancha.

## 6.1 Descripción del prototipo

El prototipo desarrollado realiza dos funcionalidades:

- **Selección de la instancia del problema**: Al correr la aplicación se inicia con una ventana, la cual se aprecia en la figura 6.1, donde se elije la instancia del problema a resolver, luego se lee el archivo benchmark que contiene el número de periodos, mínimo y máximo de cursos, mínimo y máximo de créditos, la lista de cursos, la lista de créditos y la lista de prerrequisitos de cada curso. La estructura de estos archivos se encuentran en la CSPLib.

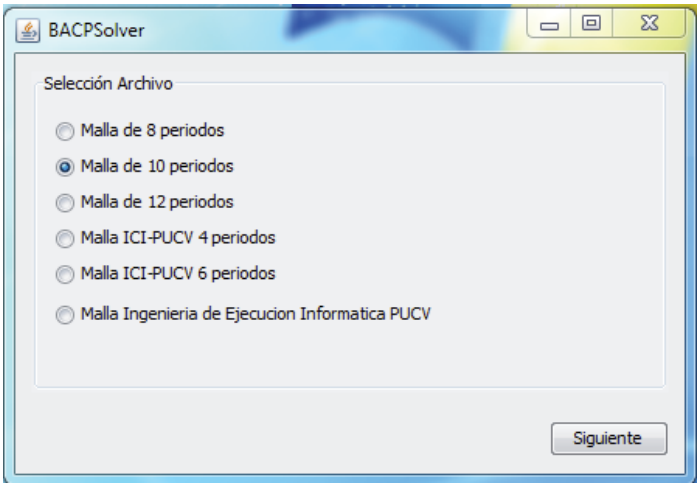


**Figura 6.1 Ventana inicial del prototipo**

- **Creación de la malla curricular balanceada:** En la figura 6.2 se muestra la ventana donde se seleccionan los parámetros a utilizar para el modelo del SMPH y además las restricciones blandas del problema. La solución que entregue se puede ver en la figura 6.3 donde se aprecia cada periodo con sus asignaturas asignadas, el número total de créditos que este posee y entrega el tiempo que demora el algoritmo en ejecutarse.

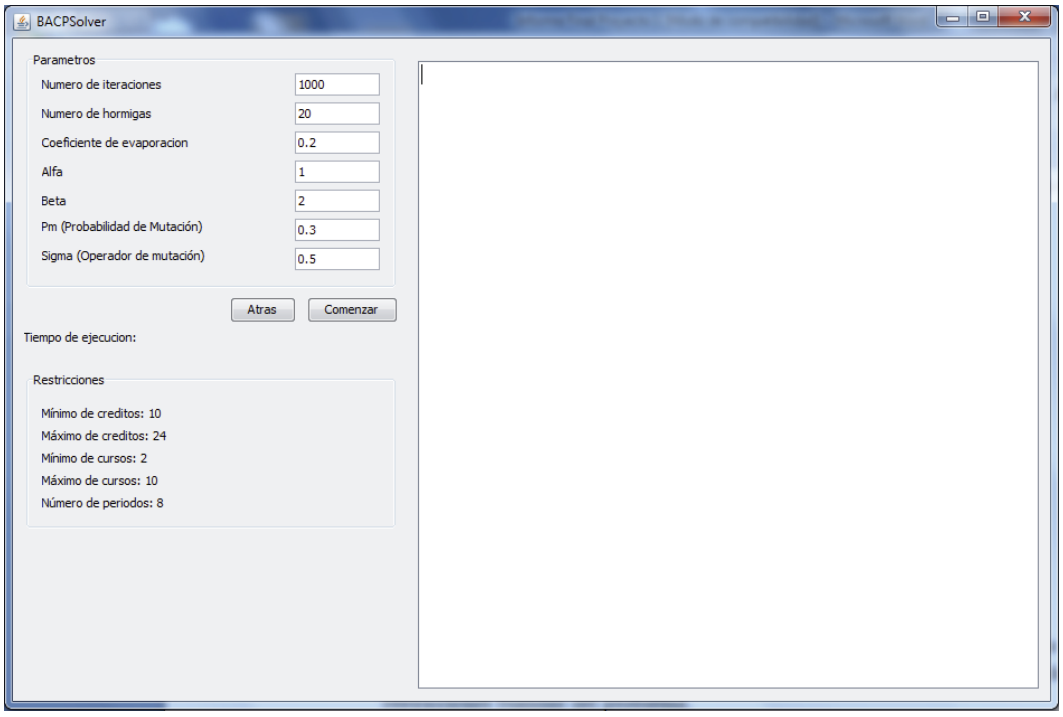


**Figura 6.2: Ventana de selección de parámetros y ejecución del algoritmo**

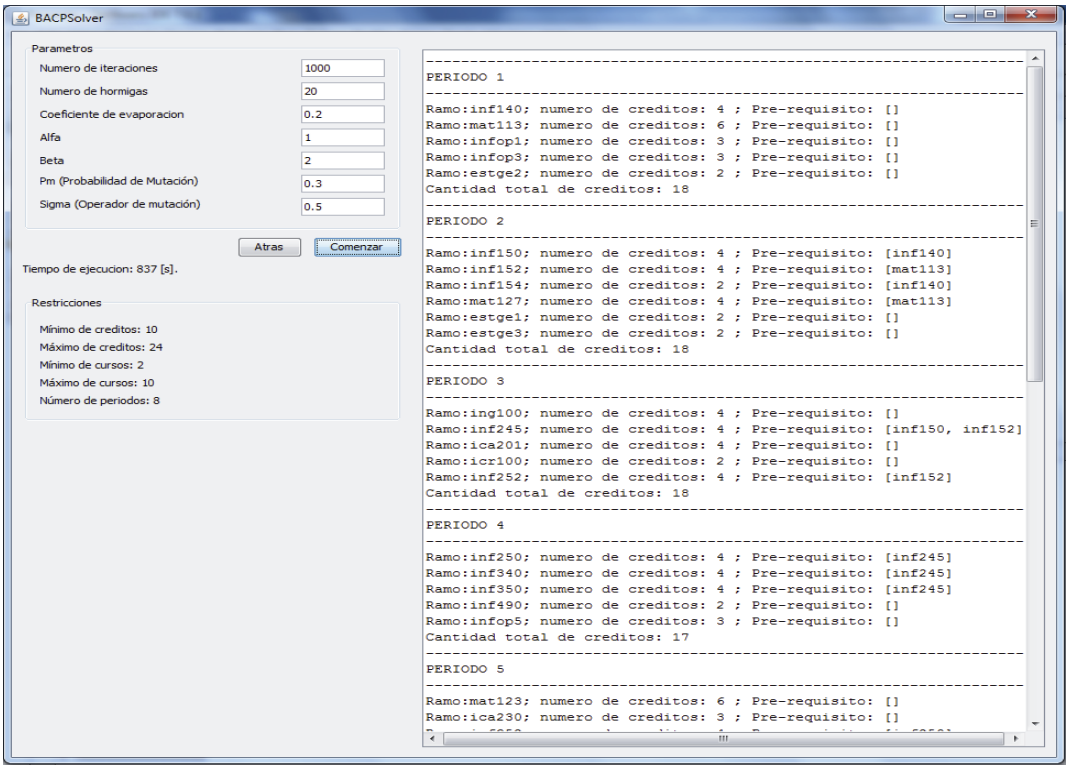


**Figura 6.3: Formato de salida de la solución encontrada**

## 6.2 Resultados del prototipo

Se realizaron 50 pruebas tanto para instancias reales como de prueba. Los parámetros utilizados en cada uno de los experimentos fueron los siguientes:

**Tabla 6.1: Parámetros del modelo SMPH utilizado en los experimentos**

| Parámetro | Valor |
|---|---|
| Número de iteraciones | 1000 |
| Número de hormigas | 8 |
| Coeficiente de evaporación | 0,2 |
| α | 1 |
| β | 2 |
| Probabilidad de mutación (Pm) | 30% |
| Operador de mutación | 0,5 |

A continuación se muestran los resultados obtenidos para cada uno de los casos estudiados.

### 6.2.1 Instancias de prueba

Las instancias de prueba sirven para medir el rendimiento del algoritmo y tener un punto de comparación con otros trabajos realizados sobre la resolución del BACP. Existen tres instancias del problema las cuales serán analizadas de manera independiente.

#### 6.2.1.1 BACP8

El bacp8 corresponde a la instancia pequeña del problema la cual posee 46 asignaturas que deben ser distribuidas en 8 periodos académicos. En este caso se logra satisfacer tanto las restricciones duras como las restricciones blandas en la totalidad de las pruebas. En la figura 6.4 se muestra un gráfico circular correspondiente a la calidad de las soluciones encontradas por el algoritmo. De un total de 50 pruebas en el 68% se logró encontrar la solución óptima conocida para este problema y en un 32% se alcanzaron soluciones validas de 18 créditos, pero que no corresponden al valor óptimo. Para esta instancia del problema, el algoritmo fue muy eficaz, ya que se satisfacen siempre todas las restricciones, tanto blandas como duras, y se llega a la mejor solución posible, en términos de calidad.

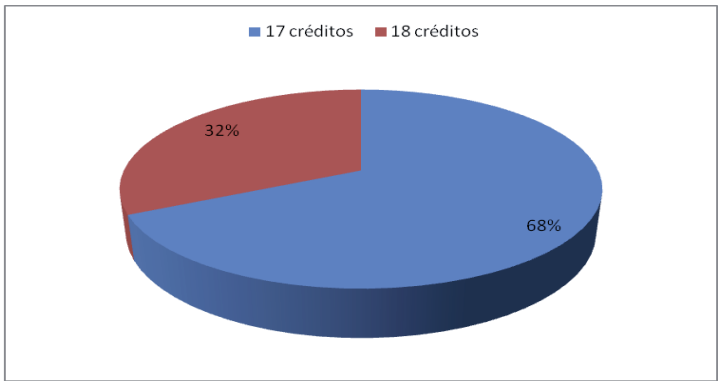


**Figura 6.4: Frecuencia con que se encontraron las distintas soluciones para el bacp8.**

De los 50 experimentos, se tomó una muestra aleatoria de 10 ejecuciones del algoritmo, donde se puede comparar la mejor y la peor solución. Los parámetros utilizados para medir que una solución es mejor que otra son: calidad de la solución y tiempo de convergencia. En la figura 6.5 se muestra la evolución que presenta la calidad de las mejores soluciones globales por iteración entre la mejor y la peor solución.

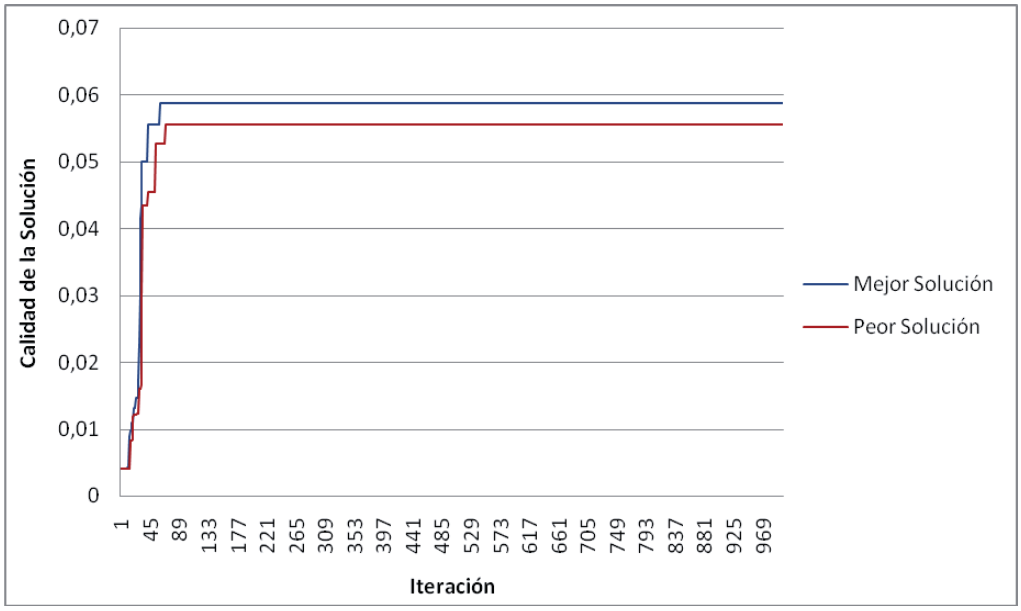


**Figura 6.5: Calidad de solución por número de iteración para el bacp8.**

Aunque ambas soluciones son bastante buenas, ya que satisfacen todas sus restricciones y en el caso de la Mejor Solución alcanza el óptimo bastante rápido (60 iteraciones). En el caso de la Peor Solución la calidad alcanzada no logra el óptimo, sin embargo es muy cercano y converge rápidamente, aproximadamente en las primeras 70 iteraciones. En la tabla 6.2 se realiza nuevamente la comparación entre la mejor y la peor solución, sin embargo, esta vez se utiliza la cantidad de créditos máxima alcanzada, es decir, la carga académica mayor entre los 8 periodos, por el tiempo en el que se logró encontrar esa solución. Aquí se demuestra que el rendimiento del algoritmo es efectivo, ya que converge al

óptimo en 1,378 segundos mientras que la peor solución también alcanza una malla balanceada y el tiempo es de 1,457 segundos.

**Tabla 6.2: Tiempo en que se alcanzan las soluciones en el bacp8**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| **Créditos** | **t [Seg]** | **Créditos** | **t [Seg]** |
| 22 | 0,733 | 23 | 0,714 |
| 20 | 0,788 | 22 | 0,888 |
| 18 | 1,003 | 19 | 1,15 |
| 17 | 1,378 | 18 | 1,457 |

### 6.2.1.2 BACP10

Esta segunda instancia posee 42 asignaturas, las cuales deben ser repartidas en 10 periodos académicos, por lo que la cantidad de créditos mayor (máxima carga académica) es menor que en el caso del bacp8. Al ser menor la complejidad de esta instancia, el comportamiento del algoritmo se ve beneficiado, ya que la solución óptima del problema corresponde a 14 créditos, la cual es encontrada en el 70% de las ejecuciones. En un 30% de las pruebas realizadas se logra una solución de 15 créditos, la cual satisface todas las restricciones. Al igual que en el bacp8, el algoritmo no llega a soluciones invalidas.

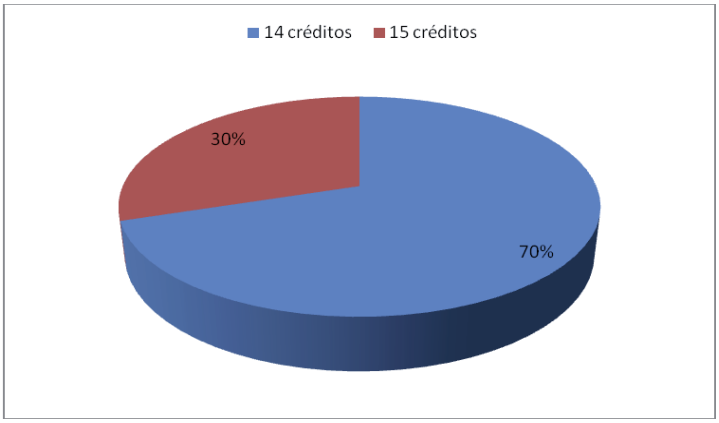


**Figura 6.6: Frecuencia con que se encontraron las distintas soluciones para el bacp10**

Se realizó la comparación entre la Mejor y la Peor Solución midiendo calidad de solución versus el número de iteración, la cual se gráfica en la figura 6.7, donde nuevamente los resultados fueron positivos desde el punto de vista de la satisfacción de restricciones y la búsqueda de la solución óptima. En 68 iteraciones se consigue el valor óptimo para esta instancia, mientras que la peor solución tarda 97 iteraciones en lograr una solución válida y cercana a ese valor óptimo.

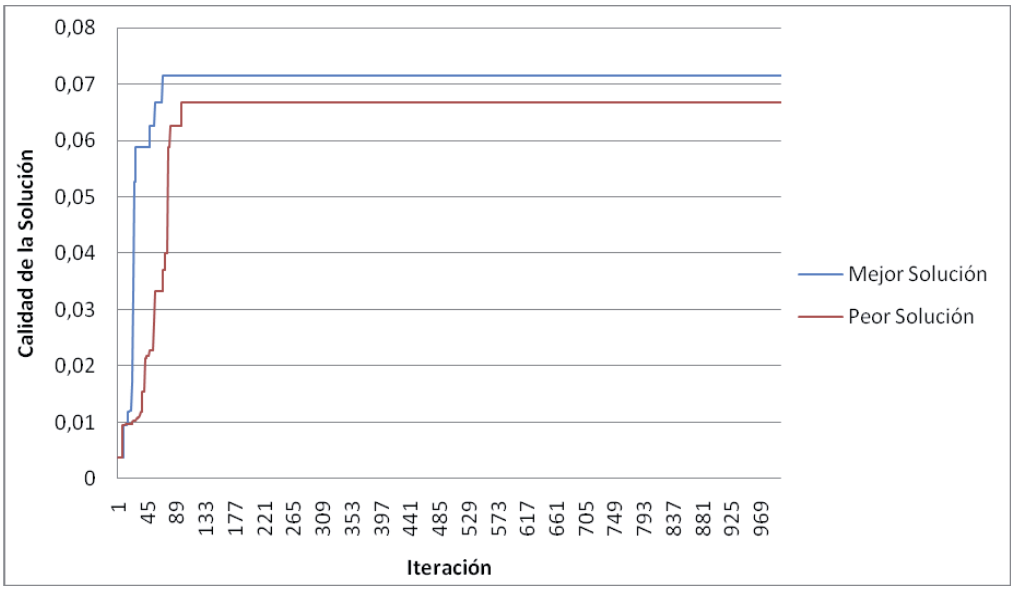


**Figura 6.7: Calidad de solución por número de iteración para bacp10**

El tiempo en el que alcanza la solución óptima es de 1,528 segundos y en la peor ejecución llegó a un máximo de 15 créditos en 2,322 segundos.

**Tabla 6.3: Tiempo en que se alcanzan las soluciones para bacp10**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| **Créditos** | **t [Seg]** | **Créditos** | **t [Seg]** |
| 17 | 0,676 | 25 | 1,512 |
| 16 | 1,121 | 17 | 1,856 |
| 15 | 1,289 | 16 | 1,945 |
| 14 | 1,528 | 15 | 2,322 |

### 6.2.1.3 BACP12

La instancia de 12 periodos es la más compleja ya que cuenta con 66 asignaturas, por lo que el tiempo computacional requerido para ejecutarse el algoritmo es mayor que en los casos anteriores. Aunque aumenta la cantidad de soluciones distintas, o sea, se generan soluciones tanto que violan restricciones blandas hasta soluciones óptimas, no se construyen soluciones inválidas que violen las restricciones duras del problema.

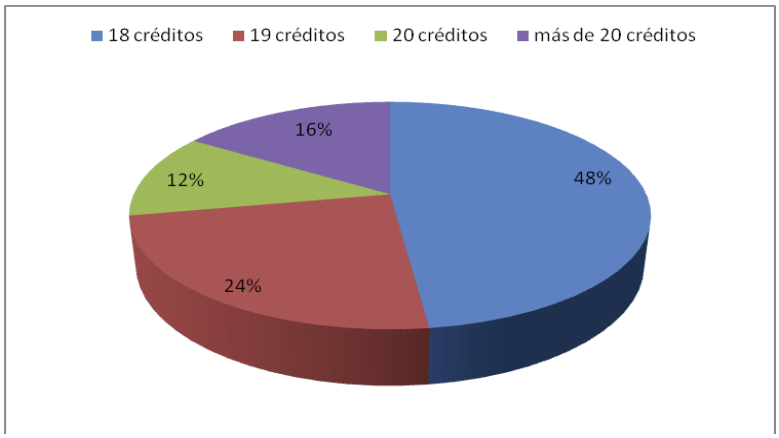


**Figura 6.8: Frecuencia con que se encontraron las distintas soluciones para bacp12**

En la figura 6.8 se puede apreciar como las soluciones con 18 y 19 créditos presentan la mayor cantidad de reiteraciones, 48% y 24% respectivamente. El valor encontrado con mayor frecuencia corresponde a 18 créditos, mientras que el porcentaje de soluciones inválidas es de 0%. En un 16% de las pruebas para esta instancia se encontraron soluciones que superan los 20 créditos. Esto ocurre al presentarse una mayor complejidad, cosa que no ocurre en ninguna de las otras instancias.

Se tomaron 10 de las 50 pruebas y se analizaron la mejor y la peor solución. En cuanto a la rapidez de convergencia, nuevamente se comprueba la eficiencia del algoritmo, ya que alcanza rápidamente una solución válida. La peor solución llega a 0,05 (20 créditos), se logra en una oportunidad y tardó hasta la iteración 160 en ser alcanzada. Por otro lado, la mejor solución tardó 80 iteraciones en lograr una calidad de solución de 0,055 (18 créditos).

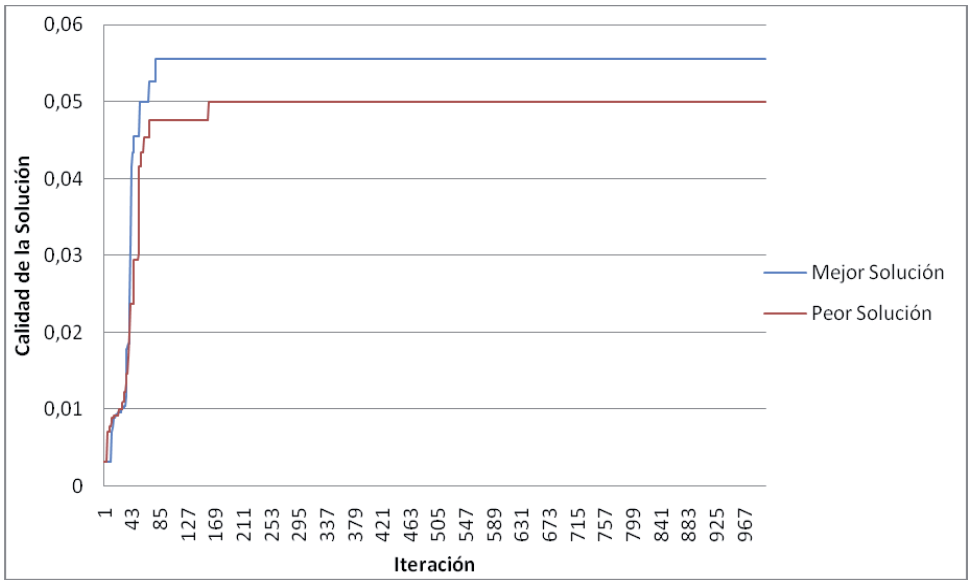


**Figura 6.9: Calidad de solución por número de iteración para bacp12**

Por último, se muestra la comparación en tiempo de la Mejor y la Peor solución en la tabla a continuación.

**Tabla 6.4: Tiempo en que se alcanzan las soluciones para bacp12**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| **Créditos** | **t [Seg]** | **Créditos** | **t [Seg]** |
| 22 | 3,463 | 23 | 4,392 |
| 20 | 4,064 | 22 | 4,668 |
| 19 | 4,994 | 21 | 5,312 |
| 18 | 5,648 | 20 | 11,636 |

La mejor solución alcanza los 18 créditos en 5,648 segundos mientras que la peor solución solo logra 20 créditos en 11,636 segundos. A pesar de que tarda más que las otras dos instancias, el tiempo sigue siendo bastante pequeño lo que demuestra que el algoritmo es efectivo y logra buenas soluciones para las tres instancias de prueba.

### 6.2.2 Instancias reales

Para evaluar las instancias reales se crearon tres archivos benchmark utilizando el mismo formato que en las instancias de prueba. Las carreras a ser evaluadas corresponden a Ingeniería en Ejecución Informática (8 periodos) e Ingeniería Civil Informática de la Pontificia Universidad Católica de Valparaíso (12 periodos), e Ingeniería Informática de la Universidad de Playa Ancha (10 periodos).

#### 6.2.2.1 Ingeniería en Ejecución Informática de la PUCV

En el análisis de las mallas generadas para Ingeniería en ejecución Informática (INF) se utilizaron 34 asignaturas a ser distribuidas en 8 periodos académicos, de las cuales se encuentran 27 asignaturas obligatorias, 2 ramos de Estudios Generales (Cultura Religiosa I y II) y 5 asignaturas optativas. De un total de 50 pruebas los resultados fueron los siguientes.

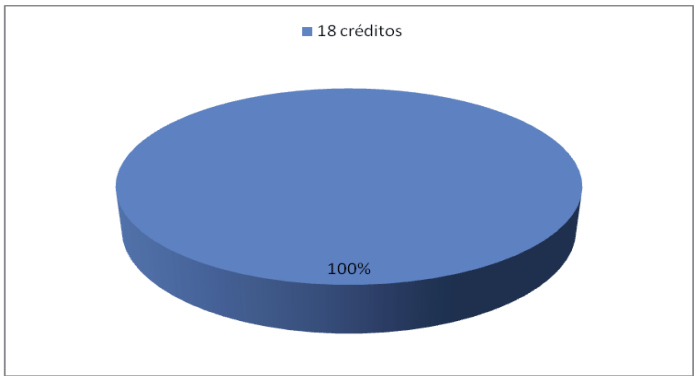


**Figura 6.10: Frecuencia con que se encontraron las distintas soluciones para INF**

En la figura 6.10 se puede apreciar la eficiencia del algoritmo al encontrar en un 100% de las ejecuciones el óptimo, el cual es menor a los 20 créditos que tiene la malla original de la carrera[7]. Al igual que en los casos anteriores, no se generan soluciones invalidas. Por otro lado, de una muestra tomada con 10 ejecuciones, se realizó una comparación entre la mejor solución obtenida y la peor. En la figura 6.11 se muestra como ambas logran el óptimo, sin embargo la diferencia viene dada por la rapidez de convergencia. En el caso de la peor solución le tomó un número mayor de iteraciones para converger.

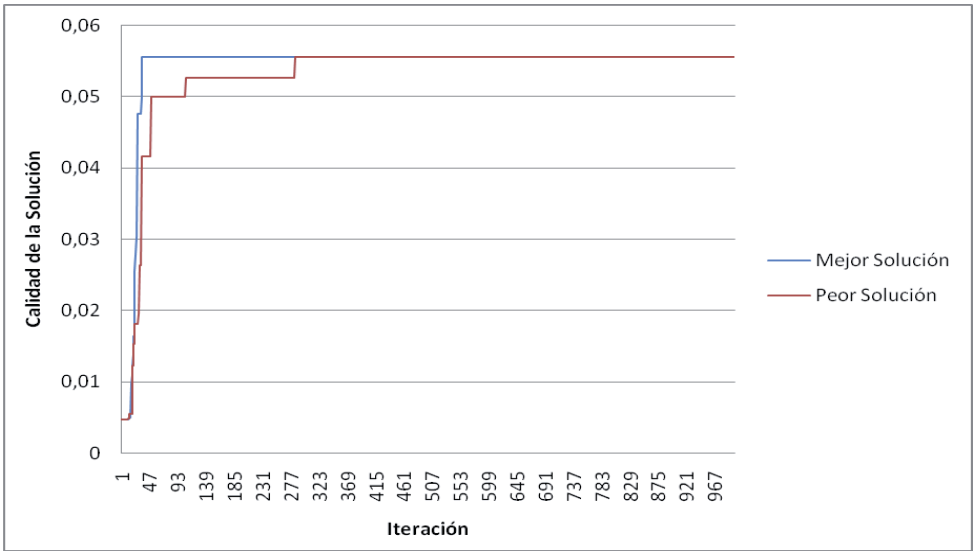


**Figura 6.11: Calidad de solución por iteración para INF**

A continuación se presenta una tabla con los resultados obtenidos midiendo el tiempo en segundos que tarda el algoritmo en llegar a su solución. La mejor solución tarda 0,479 segundos en alcanzar el óptimo, mientras que la peor solución tardó 3,527 segundos. Ambas soluciones satisfacen en su totalidad las restricciones y el tiempo sigue siendo pequeño.

**Tabla 6.5: Tiempo en que se alcanzan las soluciones para INF**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| **Créditos** | **t [Seg]** | **Créditos** | **t [Seg]** |
| 21 | 0,380 | 20 | 0,643 |
| 20 | 0,467 | 19 | 1,319 |
| 18 | 0,479 | 18 | 3,527 |

### 6.2.2.2 Ingeniería Civil Informática de la PUCV

La carrera de Ingeniería Civil Informática de la PUCV (ICI) tiene 53 asignaturas, de las cuales 49 son obligatorias y 4 optativas. En esta instancia no se consideran las asignaturas

[7] Sin considerar 3 asignaturas de estudios generales.

de estudios generales. Al igual que en la instancia anterior INF, se realizaron 50 experimentos que arrojaron los resultados mostrados en el gráfico circular de la figura 6.12.

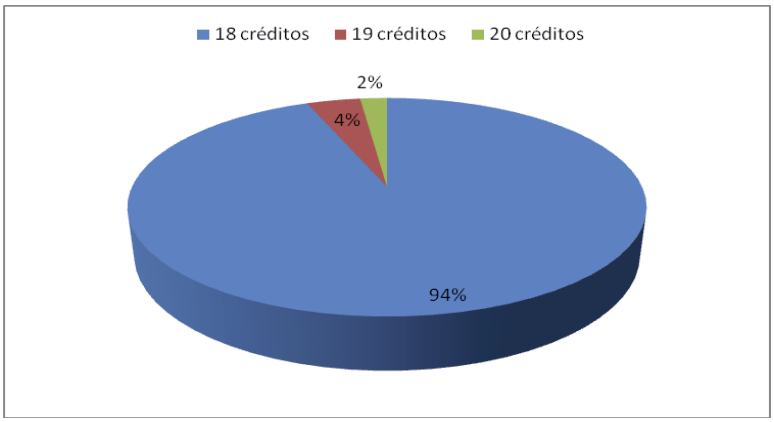


**Figura 6.12: Porcentaje de soluciones encontradas para ICI**

En el 94% de los experimentos se llegó a 18 créditos, que es el mismo valor de la malla original de esta carrera. En dos ocasiones genera una malla con 19 créditos y en el 2% (1 oportunidad) se llegó a una solución de 20 créditos.

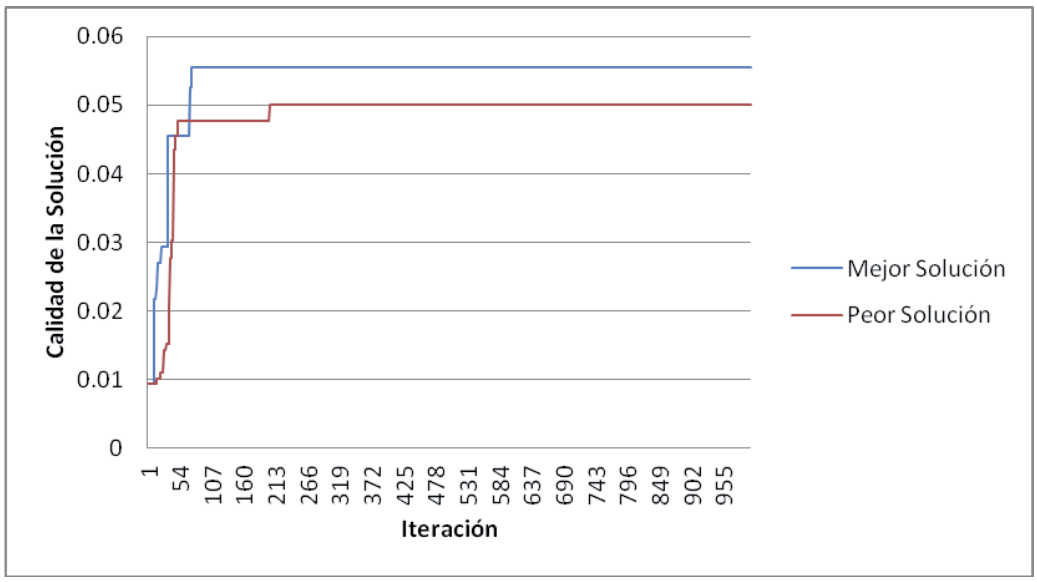


**Figura 6.13: Calidad de solución por iteración en ICI**

De las 50 ejecuciones, se seleccionaron aleatoriamente 10 soluciones donde en su mayoría lograron encontrar una solución óptima. En el caso de la Mejor Solución, en la iteración 73 alcanza el óptimo rápidamente mientras que en la Peor Solución demoró 202 iteraciones a una calidad más baja que el valor óptimo. En cuanto al tiempo de convergencia, la mejor solución alcanza su óptimo en 2,806 segundos mientras que la peor solución lo logra en 8,185 segundos. Para esta instancia real, no se generaron soluciones inválidas.

**Tabla 6.6: Tiempo en que se alcanzan las soluciones para ICI**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| Créditos | t [Seg] | Créditos | t [Seg] |
| 22 | 1,375 | 23 | 1,925 |
| 19 | 2,699 | 22 | 2,053 |
| 18 | 2,806 | 21 | 2,22 |
| | | 20 | 8,185 |

### 6.2.2.3 Ingeniería Informática de la UPLA

La última instancia real analizada fue para la carrera de Ingeniería Informática de la Universidad de Playa Ancha, la cual tiene una duración de 5 años, por lo tanto se deben asignar las 49 asignaturas entre 10 periodos académicos. Se realizaron 50 experimentos obteniendo los siguientes resultados.

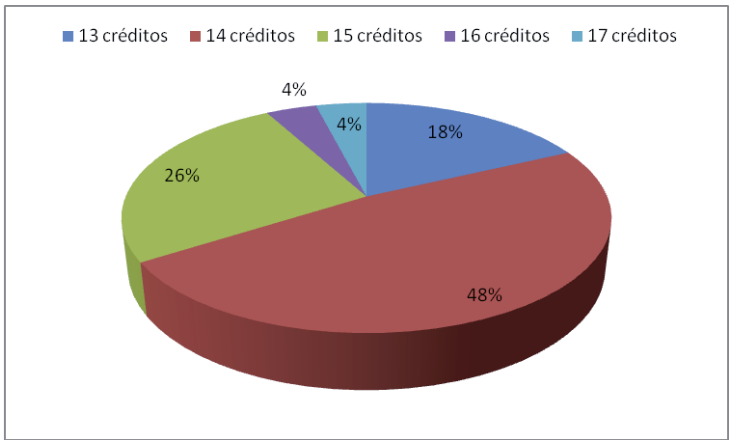


**Figura 6.14: Porcentaje de soluciones encontradas para UPLA**

En esta instancia del problema, a diferencia de las anteriores, se registro una variación más grande entre las soluciones encontradas; 13, 14, 15, 16 y hasta 17 créditos fueron las máximas cargas académicas encontradas. El óptimo fue de 13 créditos, pero solo fue encontrado en un 18%. La moda fue de 14 créditos con un 48%. Luego la solución de 15 créditos con un 26% y en dos ocasiones se encontró soluciones de 16 y 17 créditos. No se generaron soluciones inválidas.

Seleccionando aleatoriamente 10 soluciones de las 50 registradas, se realizó una comparación entre la mejor y la peor solución encontrada, la cual se muestra en la figura 6.15 y en la tabla 6.7. Primero, analizando la calidad de solución por iteración y luego la evolución medida en tiempo de las soluciones.

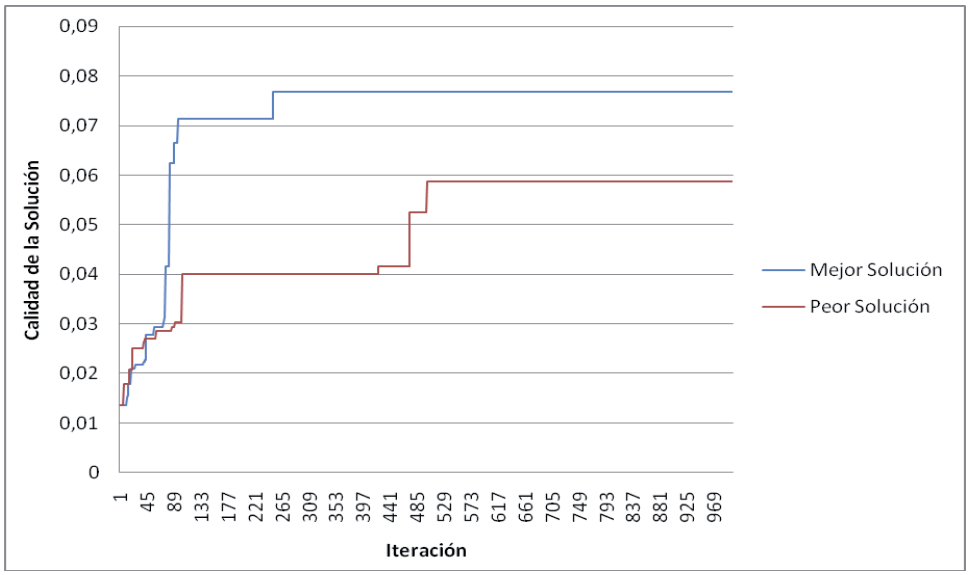


**Figura 6.15: Calidad de Solución por iteración para UPLA**

La mejor solución converge en 6,646 segundos y alrededor de la iteración 250 a una solución óptima que incluso mejora la malla curricular oficial para esta carrera, mientras que la peor solución se aleja bastante de esta cifra, ya que tarda 13,325 segundos en alcanzar una solución de 17 créditos y converger en la iteración 500.

**Tabla 6.7: Tiempo en que se alcanza la solución para UPLA**

| Mejor Solución | | Peor Solución | |
|---|---|---|---|
| **Créditos** | **t [Seg]** | **Créditos** | **t [Seg]** |
| 16 | 2,302 | 25 | 2,900 |
| 15 | 2,512 | 24 | 11,340 |
| 14 | 2,668 | 19 | 12,634 |
| 13 | 6,646 | 17 | 13,327 |

En la tabla 6.8 se resumen los resultados para las 6 instancias utilizando las muestras anteriores (las 10 ejecuciones). En la tabla se pueden ver los siguientes datos:

- Instancia del problema.
- Mejor resultado obtenido.
- Peor resultado obtenido.
- La media aritmética.
- La desviación estándar.
- Valor óptimo del problema para las instancias de prueba y valor de la malla real en el caso de las instancias reales.

**Tabla 6.8: Resultados por instancia**

| Instancia | Mejor | Peor | Media | σ | Moda | Óptimo |
|---|---|---|---|---|---|---|
| **Bacp8** | 17 | 18 | 17,1 | 0,316 | 17 | 17 |
| **Bacp10** | 14 | 15 | 14,1 | 0,316 | 14 | 14 |
| **Bacp12** | 18 | 20 | 18,5 | 0,707 | 18 | 18[8] |
| **INF** | 18 | 18 | 18,0 | 0,000 | 18 | 20 |
| **ICI** | 18 | 20 | 18,2 | 0,632 | 18 | 18 |
| **UPLA** | 13 | 17 | 14,3 | 1,159 | 14 | 14 |

En la tabla 6.8 se demuestra que el algoritmo tuvo un comportamiento positivo en términos de resultados logrados, ya que para bacp8 y bacp10 logra igualar el óptimo conocido. En el caso del bacp12 no se ha demostrado que 18 créditos es el óptimo, pero aun así es una buena solución. Además, la moda para las tres instancias mencionadas es equivalente con el óptimo y también con la mejor solución. Para los casos reales también se lograron buenos resultados. En la instancia INF, la malla original tiene una máxima carga académica de 20 créditos, mientras que el algoritmo logró llegar a una máxima de 18. En la malla ICI se logró igualar el óptimo y para la UPLA se logró mejorar los 14 créditos llegando a una máxima carga de 13. Sin embargo, en este último caso, la tendencia fue de 14 créditos debido a la complejidad mayor en contraste al resto de las instancias.

En la figura 6.16 se muestra la distribución de créditos en las distintas instancias del problema analizadas. Se puede apreciar como para casi todas las instancias, menos la UPLA, el valor óptimo es la tendencia por una amplia mayoría.

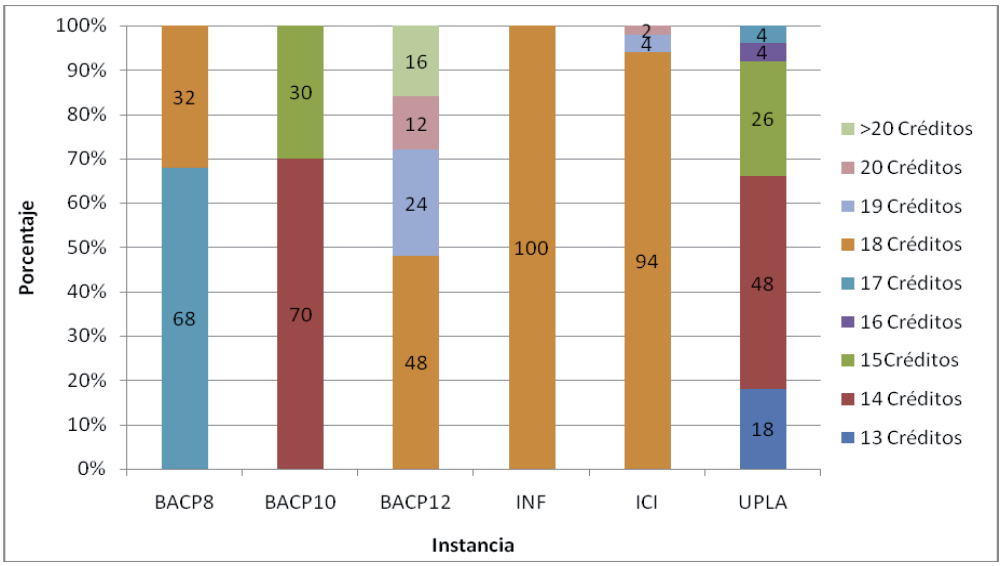


**Figura 6.16: Distribución porcentual de los resultados obtenidos en cada instancia**

[8] Este valor no se ha demostrado que es el óptimo para el problema. Se trata de la mejor solución encontrada en este trabajo.

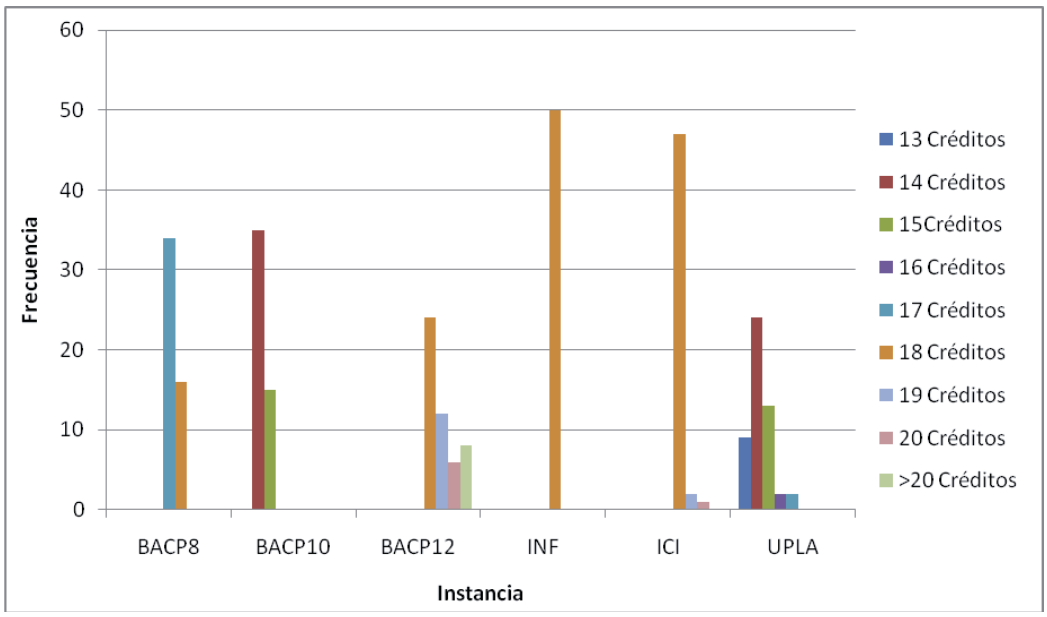


**Figura 6.17: Distribución de los resultados obtenidos en cada instancia**

Finalmente, la figura 6.17 muestra otra gráfica resumen de la carga academica maxima obtenida en cada una de las instancias y con que frecuencia fueron obtenidas. Esta vez, medido en frecuencia con que se obtuvo y no en porcentaje.

## 6.3 Resultados utilizando búsqueda local basada en movimientos de vecindario

A continuación se presentan los resultados obtenidos por el algoritmo SMPH al utilizar los movimientos de vecindario descritos en la sección 5.5. Se toman en consideración los 5 primeros tipos de movimiento, ya que el *movimiento tipo 6* al ser el que generó los mejores resultados fue el utilizado para el análisis de estos (ver sección 6.2).

### 6.3.1 Movimiento tipo 1

El movimiento tipo 1 consiste en seleccionar el 10% de los cursos (tomados aleatoriamente) y elegir el que viola más restricciones blandas y moverlo a un periodo que viole la menor cantidad de restricciones blandas. Los resultados luego de 10 ejecuciones arrojaron los resultados mostrados a continuación.

**Tabla 6.9: Resultados obtenidos utilizando movimiento de vecindario tipo 1**

| | Bacp8 | Bacp10 | Bacp12 | INF | ICI | UPLA |
|---|---|---|---|---|---|---|
| **Mejor** | 17 | 15 | 18 | 18 | 18 | 14 |
| **Media** | 17,9 | 15 | 19,7 | 18,4 | 19,3 | 14,6 |
| **Peor** | 19 | 15 | 23 | 19 | 21 | 15 |
| **σ** | 0,567 | 0 | 1,33 | 0,516 | 0,94 | 0,516 |
| **Óptimo** | 17 | 14 | 18 | 18 | 18 | 13 |

El movimiento tipo 1 logró encontrar valores óptimos en las instancias bacp8, bacp12, INF e ICI. Sin embargo, la frecuencia de óptimos locales encontrados es baja si se compara con el movimiento tipo 6 utilizado en el análisis de los resultados de la sección 6.2. En Bacp12 al tener mayor complejidad la calidad de la solución se aleja y la desviación estándar aumenta.

### 6.3.2 Movimiento tipo 2

Consiste en elegir dos cursos aleatoriamente e intercambiar periodos. Se realizaron 10 pruebas donde se encontraron los siguientes resultados.

**Tabla 6.10: Resultados obtenidos utilizando movimiento de vecindario tipo 2**

| | Bacp8 | Bacp10 | Bacp12 | INF | ICI | UPLA |
|---|---|---|---|---|---|---|
| **Mejor** | 17 | 14 | 19 | 18 | 18 | 14 |
| **Media** | 17,7 | 14,9 | 24 | 18,7 | 19,9 | 15,5 |
| **Peor** | 18 | 15 | 52 | 21 | 23 | 19 |
| **σ** | 0,483 | 0,316 | 9,98 | 0,94 | 1,52 | 1,50 |
| **Óptimo** | 17 | 14 | 18 | 18 | 18 | 13 |

En 4 de las 6 instancias se logra encontrar el óptimo. Sin embargo, al igual que en el caso anterior se puede ver en la media que las soluciones están más cercanas a 18 créditos que de 17 créditos. Instancias como INF e ICI alcanzan óptimos pero sus peores soluciones tienden a alejarse cada vez más en la medida que aumente la complejidad del problema. En UPLA y Bacp12 no se logra alcanzar soluciones óptimas y las peores soluciones se alejan demasiado, razón por la cual este movimiento de vecindario no es efectivo como el tipo 6.

### 6.3.2 Movimiento tipo 3

Este movimiento de vecindad consiste en elegir un curso al azar y moverlo a otro periodo elegido aleatoriamente. Los resultados se muestran a continuación.

**Tabla 6.11: Resultados obtenidos utilizando movimiento de vecindario tipo 3**

| | Bacp8 | Bacp10 | Bacp12 | INF | ICI | UPLA |
|---|---|---|---|---|---|---|
| **Mejor** | 17 | 15 | 19 | 18 | 18 | 14 |
| **Media** | 17,7 | 15,1 | 20,1 | 18.7 | 19,2 | 14,8 |
| **Peor** | 18 | 16 | 22 | 19 | 21 | 16 |
| **σ** | 0,483 | 0,316 | 0,87 | 0,483 | 1,135 | 0,788 |
| **Óptimo** | 17 | 14 | 18 | 18 | 18 | 13 |

En 3 de las 6 instancias no se logra igualar el rendimiento del algoritmo al utilizar el movimiento de vecindario tipo 6, ya que los valores óptimos no son logrados y en las instancias que si se logra, la frecuencia es la mínima.

### 6.3.3 Movimiento tipo 4

Se seleccionan dos periodos al azar y se moverán todos los cursos de un periodo al otro. Los resultados fueron los siguientes.

**Tabla 6.12: Resultados obtenidos utilizando movimiento de vecindario tipo 4**

| | Bacp8 | Bacp10 | Bacp12 | INF | ICI | UPLA |
|---|---|---|---|---|---|---|
| **Mejor** | 18 | 14 | 19 | 18 | 18 | 14 |
| **Media** | 18,2 | 14,9 | 21,4 | 18,8 | 19,6 | 14,6 |
| **Peor** | 19 | 15 | 25 | 21 | 22 | 16 |
| **σ** | 0,421 | 0,316 | 2,366 | 0,918 | 1,349 | 0,699 |
| **Óptimo** | 17 | 14 | 18 | 18 | 18 | 13 |

Al igual que los movimientos anteriores, se logran las soluciones óptimas esporádicamente y no para todas las instancias. Las instancias con menos complejidad alcanzan la mejor solución y las más complejas se acercan.

### 6.3.4 Movimiento tipo 5

El último movimiento a analizar consiste en mover un periodo eligiendo 2 periodos i y j al azar (i < j), luego mover todos los cursos del periodo i al periodo j, los cursos del periodo j al j-1 hasta que los del i+1 se muevan al i. Los resultados fueron los siguientes.

**Tabla 6.13: Resultados obtenidos utilizando movimiento de vecindario tipo 5**

| | Bacp8 | Bacp10 | Bacp12 | INF | ICI | UPLA |
|---|---|---|---|---|---|---|
| **Mejor** | 17 | 14 | 19 | 18 | 19 | 14 |
| **Media** | 17,8 | 15 | 20,6 | 18,7 | 19,7 | 15,7 |
| **Peor** | 18 | 16 | 23 | 19 | 21 | 17 |
| **σ** | 0,421 | 0,471 | 1,349 | 0,483 | 1,159 | 1,159 |
| **Óptimo** | 17 | 14 | 18 | 18 | 18 | 13 |

Solo en 2 instancias se logra el óptimo, ya que se dificulta mejorar las soluciones debido a las relaciones de precedencia. Aún así, los resultados son bastante cercanos y válidos. Pero solo el movimiento tipo 6 es finalmente el que logra alcanzar soluciones óptimas para todas las instancias del problema. Tanto las reales como las de prueba.

# 7 Conclusiones

En este trabajo de investigación se dio una propuesta de solución al problema del balanceo de mallas curriculares. Para lograr esto, primero se investigó y definió la problemática planteada, y se reutilizó un modelo de programación lineal entera que se consideró adecuado para captar la esencia del BACP. Se trazaron además los objetivos esperados para el trabajo. Una vez que se tuvo claro el problema a resolver se eligió la técnica que ayude a resolverlo, esta fue la optimización basada en colonia de hormigas, dado que el BACP es un problema de optimización combinatorio complejo, la investigación en los últimas dos décadas ha centrado su atención en las metaheurísticas, y una que ha logrado éxito en este campo es la OCH.

Se definieron los conceptos más importantes relacionados a el problema, la técnica y también al área de optimización en sí, más específicamente a la optimización combinatoria para lograr comprender de manera más eficiente todo lo que implicaban los modelos de OCH. También se estudio el estado del arte donde se observa que la técnica elegida nunca ha sido aplicada al BACP, por lo que puede ser un aporte, ya que se lograron buenos resultados, ya sea en la calidad de las soluciones, como en tiempo y esfuerzo computacional.

Se estudiaron 5 modelos existentes de OCH, siendo el Sistema de la Mejor-Peor Hormiga el elegido para la resolución del problema. A pesar de que cualquiera de los modelos era potencialmente factible para lograrlo, se prefirió probar con SMPH ya que [4] demuestra que es un algoritmo robusto y que entrega las mejores soluciones en circunstancias similares comparándolo con otros enfoques de OCH y con un ajuste inadecuado de sus parámetros para el problema del TSP (Problema del vendedor viajero).

Se realizó una propuesta de solución detallando mediante pseudocódigo la estructura general del algoritmo con sus funciones y procedimientos elementales mas las formulas necesarias para llevarlo a cabo. Luego se llevo a la implementación mediante un prototipo que resuelve las instancias benchmark de la CSPLib y tres instancias reales obteniendo excelentes resultados en términos de calidad de la solución, ya que en todas las instancias se alcanza el óptimo, y en tiempo computacional, donde dependiendo de la instancia, el algoritmo es capaz de encontrar resultados óptimos en cosa de segundos. Se debe mencionar además, que este trabajo fue aceptado en dos conferencias internacionales: *IV Taller Latino iberoamericano de Investigación de Operaciones* (TLAIO4) desarrollado en Acapulco, México y en el *10th International Conference on Operations Research* (ICOR2012) desarrollado en la Habana, Cuba. Lamentablemente no se pudo asistir para presentar este trabajo de investigación.

Para trabajos futuros existen alternativas como la paralelización de la Colonia de Hormigas que ayudarían a incrementar la velocidad de los procesos y otro enfoque seria la resolución del modelo propuesto por [10] que presenta una complejidad mayor a la del BACP ya que incluye mas variantes que el problema original y la reprogramación de mallas curriculares, donde se toman las asignaturas que aún no ha cursado un alumno y se asignan los restantes en una cierta cantidad nueva de periodos.

# Referencias Bibliográficas


[1] Rubio J. y Crawford B., *"Modelado y resolución del problema de balanceo de mallas curriculares"*, VIII Congreso Chileno de Investigación Operativa, OPTIMA 2009, pp. 2-3, Chile, Octubre 7-10, 2009.

[2] Castro C. y Manzano S., "*Variable and Value Ordering When Solving Balanced Academic Curriculum Problems"*, Proceedings ERCIM WG on constraints, pp. 3-4, Chile, Octubre 2, 2001.

[3] Canudas L, "*El curriculum de estudios en la enseñanza superior"*, Revista de educación superior n°2, pp. 13-26, México, 1972.

[4] Alonso S., Cordón O., Fernández de Viana I. y Herrera F., "*La Metaheurística de Optimización basada en Colonia de Hormigas: Modelos y Nuevos Enfoques*", Optimización inteligente: técnicas de inteligencia computacional para optimización, pp. 261-314, España, 2004.

[5] Pasteels J. M., Deneubourg J. L., y Goss S., "*Self-organization mechanisms in ant societies (I): Trail recruitment to newly discovered food sources",* Experentia Supplementum, Vol. 54, pp. 155-175, 1987.

[6] Dorigo M. y Di Caro G., "*The ant Colony Optimization Metaheuristic"*, New ideas in Optimization, McGraw Hill, pp. 11-32, UK, 1999.

[7] Papadimitriou C. y Steiglitz K., "*Chapter 1: Optimization problems*", Combinatorial Optimization: algorithms and complexity, pp. 4-10, 1998.

[8] Luna F., "*Metaheurísticas avanzadas para problemas reales en las redes de telecomunicaciones",* Tesis doctoral Universidad de Málaga, pp. 19-24, España, 2008.

[9] Papadimitriou C. y Steiglitz K., "*Chapter 19: Local Search*", Combinatorial Optimization: algorithms and complexity, pp. 454-455, 1998.

[10] Chiarandini M., Di Gaspero L., Gualandi S. y Schaerf A., "*The balanced academic curriculum problem revisited",* Journal of Heuristics, pp.1-30, 2011.

[11] Hnich B., Kiziltan Z. y Walsh T., "*Modelling a Balanced Academic Curriculum Problem",* Proceedings of CP-AI-OR-2002, Vol. 2, pp. 121-131, 2002.

[12] Lambert T., Castro C., Monfroy E., Riff M. y Saubion F., "*Hybridization of Genetic Algorithms and Constraint Propagation for the BACP"*, Logic Programming, Vol. 3668, pp. 421-423, Berlin, 2005.

[13] Dorigo M., Maniezzo V. y Colorni A., "*The Ant System: Optimization by a colony cooperating agents"*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 26, pp.29-41, 1996.

[14] Abdullah S., Burke E. y McCollum B., "*An investigation of variable neighborhood search for University course timetabling"*, The 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications., pp. 413-427,United Kingdom, 2005.