Inteligencia Artificial Avanzada

Estado del Arte: *Balanced Academic Curriculum Problem (BACP)*

Felipe Chacón Ossa

31 de agosto de 2018

Resumen

En el presente, se abordará el problema de *Balanced Academic Curriculum Problem (BACP)*, de qué trata, sus orígenes, sus posibles aplicaciones y la importancia que tiene dentro del mundo universitario. Una gran variedad de técnicas han sido aplicadas en distintos modelos de este problema, como también una variante que generaliza *BACP* para abordar otro tipo de restricciones, como las preferencias de los profesores. Dentro de las técnicas que se revisarán se encuentra el modelo original de satisfacción de restricciones, un modelo híbrido de programación lineal con satisfacción de restricciones, un algoritmo de colonia de hormigas y una alternativa con algoritmos genéticos.

Finalmente, se describe un algoritmo de colonia de hormigas para obtener una solución inicial al problema, para luego utilizar un algoritmo reparador y obtener resultados basados en la desviación estándar entre los créditos de cada periodo. Se realizan experimentos en los parámetros de los algoritmos y se sacan conclusiones basándose en ellos.

1. Introducción

La asignación de cursos a través del programa de una carrera es un problema que enfrentan todas las universidades, y es el paso previo a elegir los horarios de esos cursos en cada periodo académico. La complejidad de este desafío yace en los pre-requisitos de los cursos, pues al momento de modelar y resolver el problema, se debe asegurar que esta restricción se cumple y esté representada de alguna forma eficiente en el modelo.

Balanced Academic Curriculum Problem (BACP) enfrenta esta situación y cuyas soluciones óptimas obtenidas son las asignaciones de los cursos en los periodos, entendiendo óptima como una carga académica balanceada en todos los periodos. *BACP* es parte de la CSPLib [3], fue introducido por Castro y Manzano [1] junto con las 3 instancias básicas del problema que son de 8, 10 y 12 periodos académicos, estas instancias corresponden a 3 carreras del Departamento de Informática de la Universidad Técnica Federico Santa María.

En la sección 2 se explica el problema junto a sus variables y restricciones, además de un resumen de su variante *General BACP*. En la sección 3 se revisan los avances que han habido en relación a este problema, cuáles han sido los modelos propuestos, qué técnicas se han desarrollado y las formas de resolverlo. En la sección 4 se presenta el modelo matemático del problema. La sección 5 detalla la representación de la solución. La sección 6, describe el algoritmo junto a sus procesos. En la sección 7 se encuentran los experimentos realizados variando los parámetros del algoritmo. Los resultados obtenidos en las instancias utilizadas están en la sección 8. Finalmente, en la sección 9T se encuentran las conclusiones.

2. Definición del Problema

Balanced Academic Curriculum Problem (BACP) consiste en asignar cursos a los periodos académicos del programa o malla curricular de una carrera, de forma que la carga académica total para los estudiantes sea balanceada a través de su etapa estudiantil.

La carga académica de un curso es medida por los créditos que tiene, los que representan el tiempo que debe ser dedicado al curso para aprender los contenidos que tiene. Existe un mínimo y un máximo de créditos para cada semestre, el primero es para ser considerado estudiante de tiempo completo o regular y el último para evitar una sobrecarga académica.

Junto al mínimo y máximo de créditos, también se consideran estos límites a la cantidad de cursos, existe un mínimo para mantener un avance consistente con la duración del programa de la carrera y un máximo para evitar sobrecarga académica.

Además, otra restricción importante es que los cursos pueden tener pre-requisitos que deben cumplirse, no pudiendo ser asignados si el curso pre-requisito no lo fue en un periodo anterior.

El primer modelo presentado en [1] trabaja bajo las siguientes definiciones y restricciones:

- Programa o Malla Currícular: Compuesto de periodos académicos en los cuales los cursos van asignados.
- Número de periodos: Todos los cursos deben ser asignados en algún periodo.
- Carga académica: Cada curso tiene asociado una cantidad de créditos que representan el esfuerzo y tiempo necesarios para aprender los contenidos del mismo curso.
- Pre-requisitos: Algunos cursos pueden tener como pre-requisitos otros cursos.
- Carga académica mínima: Existe una mínima cantidad de créditos por periodo para ser considerado alumno regular.
- Carga académica máxima: Existe una máxima cantidad de créditos por periodo para evitar la sobrecarga académica.
- Cantidad de cursos mínimo: Se necesita estar cursando una mínima cantidad de cursos para ser considerado alumno regular.
- Cantidad de cursos máximo: Hay una máxima cantidad de cursos que pueden ser cursados al mismo tiempo en un periodo para evitar la sobrecarga académica.

2.0.1. General BACP

Una variante del problema fue propuesta en [2]. Motivados por el trabajo en [6], en el que fueron resueltas las instancias originales e incluso se generaron nuevas instancias, fue considerado que *BACP* no refleja del todo la realidad que enfrentan las universidades, por lo que se decidió reformularlo. Tomando el problema anterior como base, ahora también se considera la inclusión de más de un programa a la vez, debido a que hay cursos que son comúnes para varias carreras. Además, también se considera las preferencias de los profesores y su disponibilidad para hacer clases en ciertos periodos, como por ejemplo, solo hacer clases en semestres pares.

Junto a la reformulación del problema que pasa a llamarse *General Balanced Academic Curriculum Problem*, 10 nuevas instancias fueron creadas provenientes de carreras de la Universidad de Udine. Siendo estas más grandes y con mayor variación respecto a las 3 originales de [3].

Estas instancias, junto con *GBACP* suponen un problema más difícil de resolver. A continuación se presenta una comparación entre las instancias:

Instancias	Periodos (años x terms)	Cursos	Programas	Cursos por programa	Cursos por periodo	Prerequ.	Pref
csplib8	8 (4 x 2)	46	1	46	5.75	33	0
csplib10	10 (5 x 2)	42	1	42	4.20	33	0
csplib12	12 (6 x 2)	66	1	66	5.50	65	0
UD1	9 (3 x 3)	307	37	34.62	3.847	1383	270
UD2	6 (2 x 3)	268	20	27.8	4.633	174	158
UD3	9 (3 x 3)	236	31	29.81	3.312	1092	198
UD4	6 (2 x 3)	139	16	25.69	4.281	188	80
UD5	6 (3 x 2)	282	31	34.32	5.72	397	162
UD6	4 (2 x 2)	264	20	27.15	6.787	70	110
UD7	9 (3 x 3)	302	37	33.89	3.766	1550	249
UD8	6 (2 x 3)	208	19	22.58	3.763	149	120
UD9	9 (3 x 3)	303	37	34.08	3.787	1541	255
UD10	6 (2 x 3)	188	19	25.07	4.178	214	110

3. Estado del Arte

3.1. Modelo de Programación Entera y Modelo Basado en Restricciones

El problema fue introducido en el año 2001 por Castro y Manzano en [1], quienes además propusieron las instancias que se utilizan como *benchmark* hasta el día de hoy en [3]. En [1], se explora un modelo de programación entera utilizado para resolver este tipo de problemas y luego se presenta uno basado en restricciones que es más eficiente que el modelo anterior.

3.1.1. Modelo de Programación Entera

En el modelo de programación entera, la representación es a través de variables binarias y se pueden ver como si fuera una matriz de 2 dimensiones, donde las filas son los cursos y las columnas los periodos:

$$x_{ij} = \begin{cases} 1 & \text{Si el curso } i \text{ es asignado al periodo } j \\ 0 & \text{en otro caso} \end{cases}$$

En cuanto a la función objetivo, se opta por minimizar la carga académica del periodo con la mayor carga, es decir:

$$\text{Min } c = \text{Max}\{c_1, \ldots, c_n\}$$

Donde c_i es la carga académica del periodo i y c es la carga académica mayor de entre todos los periodos.

En las restricciones se encuentran aquellas que dictan el mínimo y máximo, tanto de créditos como de cursos, además de otras que aseguran que los cursos sean asignados exactamente 1 vez en todo el programa de la carrera. Para la restricción de los pre-requisitos, si consideramos que el curso b tiene como pre-requisito el curso a, la restricción es de la siguiente forma:

$$x_{bj} \leq \sum_{r=1}^{j-1} x_{ar} = 1 \quad \forall j = 2, \ldots, n$$

Considerando n como la cantidad total de periodos, en la restricción se indica que el curso b no puede ser asignado si es que el curso a no lo fue en algún periodo anterior.

Este modelo, utilizando *Branch & Bound* y el software *lpsolver*, solo logra resolver instancias pequeñas y en tiempos considerables. Para una instancia de 8 periodos, el algoritmo demoró *1459.73* segundos en encontrar el óptimo. Para 10 periodos no encontró una solución óptima en las 5 horas que iteró. Finalmente, para los 12 periodos, no logró encontrar solución alguna.

3.1.2. Modelo Basado en Restricciones

En este modelo, se busca satisfacer restricciones y entregar un puntaje dada una función objetivo. Dado un set de restricciones, se encuentra una solución α, luego se añade la restricción $f < \alpha(f)$, la que provoca que en la siguiente iteración se deban encontrar soluciones mejores a las ya encontradas. En el caso de que no se encuentren soluciones que satisfagan las restricciones, el óptimo es la última solución factible encontrada.

Para el problema de *BACP*, se mantiene la función objetivo del modelo anterior y, además, se define una variable c que es una cota superior a la cantidad de carga académica máxima para cada periodo. La forma de representar la asignación de cursos es mediante un vector de N cursos por K periodos.

Para resolver este modelo, se utilizó *Oz*. Debido a cómo opera este solver, el orden de los cursos y si el vector es NxK o KxN impactan enormemente en las soluciones encontradas, desde no encontrar ninguna a demorarse unos cuantos segundos en encontrar la mejor solución.

De este último modelo, para disminuir el espacio de búsqueda, todos los cursos que tienen pre-requisitos tienen asignado un 0 en el primer periodo, ya que no pueden estar asignados en él.

3.1.3. Integración de Modelos

En [4] se explora la opción de integrar modelos de programación lineal con los basados en restricciones, de forma de suplir las desventajas de cada uno de ellos por separado. Esta integración debiese venir con un aumento en el tiempo de ejecución ya que se agregan más variables al problema, sin embargo, en ocasiones puede disminuir al tener más criterios con los que reducir el espacio de búsqueda durante las iteraciones.

El modelo lineal presentado en [4] es similar al de [1], utilizando una matriz binaria 2 dimensional para representar si un curso i fue asignado en el periodo j. Esta representación requiere de una restricción adicional para asegurar que cada curso sea asignado solo 1 vez. Las ventajas de este modelo es su facilidad para identificar la carga académica de cada periodo, pues basta con sumar las columnas de la matriz.

Por otro lado, su desventaja es la dificultad que presenta la restricción de los pre-requisitos. Este modelo es llamado *ILP* al ser resuelto por un solver de programación lineal entera, mientras que si es resuelto con métodos de modelos basados en restricciones, es llamado CP_1.

Adicional al anterior, otro modelo basado en restricciones es planteado. En él, la representación de la asignación de cursos es realizada con una matriz no binaria de 1 dimensión llamada *CURRICULUM*, donde cada columna corresponde a un curso y el valor de ella corresponde al periodo en el que se encuentra asignado. Esta representación, por si sola, asegura que cada curso estará asignado a solo 1 periodo, por lo que no se requiere una restricción adicional como ocurre con el modelo anterior. La restricción de los pre-requisitos es definida de la siguiente forma:

$$\forall (i,j)\ \epsilon \text{ prereq.} \quad \text{CURRICULUM}[i] \leq \text{CURRICULUM}[j] \tag{1}$$

Considerando que los cursos se encuentran ordenados en *CURRICULUM* respetando los pre-requisitos, la ventaja de este modelo es la restricción en (1), pues ahora solo revisa dos columnas de la matriz en vez de todas las anteriores como ocurre en el primer modelo, sin embargo, tiene como desventaja su representación, pues contar la carga académica de cada periodo implica realizar más operaciones sobre la matriz *CURRICULUM* de 1 dimensión que si fuera de 2 dimensiones. Este modelo es llamado CP_2.

Finalmente, se unen los modelos de *ILP* con el de CP_2, la restricción de pre-requisitos es cargada a la representación de CP_2 y la carga académica de cada periodo a la de *ILP*, esto beneficia un modelo con las ventajas del otro para obtener mejores soluciones y, en algunos casos, mejorar el tiempo de ejecución. CP_1 y CP_2 también son integrados de forma similar a la anterior.

La integración implica que existen 2 representaciones de la asignación de cursos en los periodos, es decir, hay redundancia en las variables y, por tanto, debe ser manejada para mantener una consistencia entre las representaciones definidas como *CURRICULUM1[cursos]* y *CURRICULUM2[cursos, periodos]*, por lo que la siguiente restricción es agregada:

$$\text{CURRICULUM1}[i] = j \leftrightarrow \text{CURRICULUM2}[i, j] = 1$$

3.1.4. Resultados

Los modelos fueron probados en [4] con las 3 instancias *benchmark* de [3]. Para resolver el modelo de *ILP* fue utilizado el solver *CPLEX*, el que encontró soluciones para las 3 instancias, a diferencia de [1] que utiliza *lpsolver*. Para resolver los CP, fue utilizado el solver de *OPL* [10].

Los mejores resultados se obtuvieron de la integración de modelos, donde destaca el de CP_1 con CP_2 que demoró una menor cantidad de tiempo en encontrar una solución óptima.

Una tabla con los detalles en tiempo de ejecución para cada modelo es mostrada a continuación:

Modelo	Resultados	8 periodos	10 periodos	12 periodos
ILP	runtime	3.45	4.23	131.30
	failures	N/A	N/A	N/A
CP_1	runtime	58.52	-	-
	failures	499336	-	-
CP_2	runtime	45.10	-	-
	failures	56766	-	-
$ILP + CP_1$	runtime	0.81	8.44	3.05
	failures	183	4445	525
$CP_1 + CP_2$	runtime	0.29	0.59	1.09
	failures	651	1736	1539

3.2. Algoritmo de la colonia de hormigas

En [8] se propone resolver el problema utilizando la metaheurística de *Ant Colony Optimization* llamada *Best-Worst Ant System (BWAS)*, la que contiene tres componentes principales:

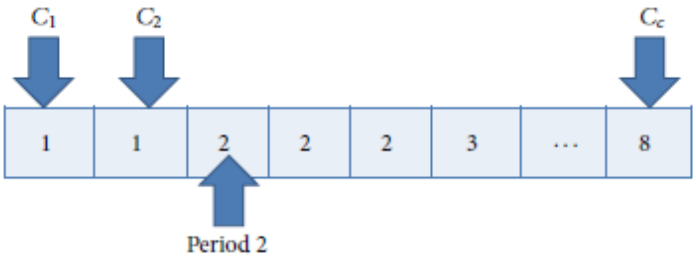
1. Actualización de las feromonas del mejor y peor trayecto: Agregar feromonas a los trayectos que están contenidos en buenas soluciones y quitar feromonas a aquellos que son parte de la peor solución.

2. Mutación de las feromonas de un trayecto: Dependiendo de una probabilidad, las feromonas de un trayecto pueden mutar añadiendo o quitando la misma cantidad que tienen para favorecer la diversidad en el proceso de búsqueda.

3. Reinicio: Cuando ocurre un estancamiento, es decir, unos pocos caminos tienen muchas feromonas y el resto una cantidad cercana a cero, se reinician las feromonas de todos los caminos a las cantidades iniciales para seguir explorarando dentro del espacio de búsqueda.

Debido a que la técnica de las hormigas es más utilizada en grafos, el problema se modela con una representación de nodos y caminos, donde cada nodo está representando la asignación de un curso a un periodo. Por simplicidad, el grafo es representado como un vector en el algoritmo,

Figura 1: Representación del grafo con un vector.



como es mostrado en la Figura 1. Las hormigas van avanzando por cada curso y deciden qué periodo asignarle al curso en el que están. Las feromonas son depositadas en la asignación de un curso c a un periodo i.

Para la disposición inicial de feromonas, se usa una solución inicial en donde la restricción de los pre-requisitos sea cumplida, sin tomar en cuenta las otras restricciones del problema.

Otro punto que se muestra, es que la función de visibilidad es variable para este problema, dando más peso a aquellas soluciones que violan menos restricciones. Definiendo $v_d(c, p_i)$ como la cantidad de restricciones violadas al asignar el curso c en el periodo i y $v_a(c, p_i)$ como la cantidad de restricciones violadas antes de la asignación del curso c, la siguiente función de visibilidad es presentada:

$$\eta(c, p_i) = \frac{1}{1 + v_d(c, p_i) - v_a(c, p_i)}$$

En el caso que ocurra una singularidad, como por ejemplo $v_d = 1$ y $v_a = 2$, el nodo es preferible al violar menos restricciones, por lo que v_a es igualado a 1.

Además, elementos de búsqueda local son agregados al algoritmo, utilizados en la mejor solución encontrada luego de que todas las hormigas siguieron un camino. En esta etapa, se eligen los cursos del periodo con la mayor carga académica y se intentan reasignar al periodo con la menor carga académica. Añadiendo otro elemento para explotar el espacio de búsqueda.

Para las instancias *benchmark*, el algoritmo, luego de correr 50 veces, encontró en su mayoría de los casos (más del 50 %) el óptimo para las instancias de 8 y 10 periodos. Para la instancia más grande de 12 periodos, el algoritmo encontró el óptimo, pero no con tanta seguridad como en los anteriores, el 42 % de las veces convergió al óptimo de 18 créditos, mientras que en el 40 % la solución fue de 19 créditos. En su mayoría, el algoritmo encontró una solución antes de las primeras 100 iteraciones.

Otras instancias creadas por los autores fueron probadas, correspondientes a 2 carreras de Ingeniería Informática de la Pontificie Universidad Católica de Valparaíso (PUCV) con 8 y 10 periodos, y a Ingeniería Informática de la Universidad de Playa Ancha (UPLA) con 10 periodos. Nuevamente el algoritmo fue ejecutado 50 veces, en los que encuentra soluciones en poco tiempo, menor a 10 segundos por lo general, siendo al menos tan buenos como las mallas curriculares reales de esas carreras. Para las instancias de la PUCV, el óptimo se alcanzó en más del 80 % de las veces. En el caso de la UPLA, solo el 20 % convergió al óptimo de 13 créditos y un 58 % a 14 créditos, que es la realidad de la malla curricular de esa carrera.

3.3. BACP con algoritmos genéticos

Debido a la efectividad de los algoritmos genéticos en distintos tipos de problema, en [9] se utiliza para abordar el *Balanced Academic Curriculum Problem*. Mientras que el modelo matemático es el mismo que en [1], la representación utilizada es un arreglo de una dimensión

del mismo tamaño que la cantidad de periodos, cuyos miembros son listas que contienen los cursos asignados a ese periodo. Se da como ejemplo una pequeña instancia de 4 periodos y 10 cursos: $I = \{\{1, 2, 3\}, \{2, 6, 10\}, \{7, 5\}, \{8, 9\}\}$.

El algoritmo genético implementado hará uso de dos mutaciones: *swap*, que cambiará 2 cursos de distintos periodos entre ellos; y *shift*, que tomará el curso de un periodo y lo reasignará en otro. Ambas mutaciones se harán tomando en cuenta las restricciones de pre-requisitos, por lo que no se podrá mutar cursos que rompan esa restricción.

Como método de inicialización de una solución, los cursos son ordenados de menor a mayor según su cantidad de pre-requisitos. Luego, son asignados en cada periodo intentando que en cada uno de ellos se tenga la misma cantidad de cursos. Finalmente, se realizan *swaps* entre los periodos luego de asignar los cursos para agregar aleatoriedad a las soluciones iniciales.

Al igual que en los modelos anteriores, la función de evaluación corresponde a la carga académica del periodo con mayor carga académica. De esta forma, se pueden comparar soluciones, siendo mejores las que tienen el puntaje más bajo.

El algoritmo genético implementado cuenta con 7 parámetros. 3 de estos parámetros son cómunes a este tipo de algoritmos, *population size (ps)*, *maximum generations (mg)* y *tournament size (ts)*. Las restantes son específicos para este problema y están pensados para dar más aleatoriedad a las soluciones encontradas, estos son *initial solution randomness intensity (is)*, *swap mutate intensity (sw)*, *shift mutate intensity (sh)*, estas 3 anteriores indican la cantidad de veces que se aplicará la mutación específicada en esa etapa. Finalmente, existe un parámetro *af* que indica un cambio en la mutación a realizar cada cierta cantidad *af* de generaciones.

Los experimentos fueron realizados en las instancias benchmark de [3] y comparados con técnicas propuestas anteriormente para abordar el *BACP*. El algoritmo genético logra resultados óptimos tanto en la instancia *BACP8* como en la *BACP10*, mientras que para *BACP12* sus resultados fueron 6 % peores que el *Tabu Search* utilizado en [7], pero iguales que en [8] y en [5]. Respecto a tiempos de ejecución, debido a que las propuestas han sido implementadas en computadores con distintas características, no es posible hacer una comparación directa. Sin embargo, para términos de esta implementación, todas las ejecuciones demoraron menos de 1 segundo.

En este último documento [9], se observa que las técnicas incompletas abundan a la hora de intentar resolver el problema de *BACP*, yendo desde *Simulated Annealing*, *Tabu Search* hasta algoritmos genéticos e incluso el de la colonia de hormigas. Técnicas que son complejas por la cantidad de parámetros que tienen y que deben configurarse según el tamaño de la instancia a resolver.

A pesar de su complejidad, las técnicas incompletas han encontrado los resultados óptimos en muy poco tiempo, en comparación a las técnicas completas que, incluso, no llegan a converger para ciertas instancias.

En cuanto a representaciones, aquellas que trabajan con matrices de 1 dimensión funcionan mejor debido a su facilidad de manejar las restricciones de pre-requisitos y de que, ya sean los periodos o los cursos, están contenidos en los índices de la matriz.

Finalmente, según los resultados mostrados en [9], *Tabu Search* ha resuelto de mejor forma *BACP* en términos de encontrar la solución óptima. Respecto al tiempo, mientras no se implementen las técnicas en una misma máquina, es difícil sacar conclusiones, pues a medida que avanza el tiempo, las características de los computadores van mejorando y, por consecuencia, el tiempo de ejecución también.

4. Modelo Matemático

El modelo propuesto es el que se puede encontrar en [1]. Explicando y definiendo sus variables, parámetros, función objetivo y restricciones.

4.1. Variables

- $$x_{ij} = \begin{cases} 1 & \text{Si el curso } i \text{ es asignado al periodo } j.\ \forall i = 1, \ldots, m\ \forall j = 1, \ldots, n \\ 0 & \text{en otro caso.} \end{cases}$$
- c_j: Carga académica total del periodo j.
- c: Carga académica máxima de todos los periodos.

Las variables x_{ij} pueden ser vistas como una matriz de 2 dimensiones, en donde las filas i corresponden a los cursos y las columnas j a los periodos. De esta forma, para calcular la carga académica de un periodo, basta con identificar en su columna los cursos que fueron asignados a ese periodo.

4.2. Notación y parámetros útiles para definir las restricciones

- m: Es la cantidad de cursos.
- n: Es la cantidad de periodos académicos. Pueden ser semestres, trimestres, etc.
- α_i: Cantidad de créditos del curso i, con $i \in \{1, \ldots, m\}$
- β: Carga académica mínima permitida por periodo.
- γ: Carga académica máxima permitida por periodo.
- δ: Cantidad mínima de cursos permitidos por periodo.
- ϵ: Cantidad máxima de cursos permitidos por periodo.

4.3. Función Objetivo

A continuación, se presenta la función objetivo con su descripción:

$$\text{Min } c = \text{Max}\{c_1, \ldots, c_n\}$$

La función objetivo es minimizar la carga académica del periodo con mayor carga académica.

4.4. Restricciones

- La carga académica de cada periodo j esta dada por:

$$c_j = \sum_{i=1}^{m} \alpha_i \cdot x_{ij}, \forall j = 1, \ldots, n$$

- Todos los cursos deben ser asignados a algún periodo j:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, \ldots, m$$

- Curso b tiene como pre-requisito el curso a, esta restricción impide que se asigne el curso b si es que no ha sido asignado a en algún periodo anterior a j:

$$x_{bj} \leq \sum_{r=1}^{j-1} x_{ar} = 1, \forall j = 2, \ldots, n$$

- La carga académica máxima de todos los periodos es asignada a c:

$$c = \text{Max}\{c_1, \ldots, c_n\} \leftrightarrow c_j \leq c, \forall j = 1, \ldots, n$$

- La carga académica del periodo j debe ser mayor o igual al mínimo establecido de β:

$$c_j \geq \beta, \forall j = 1, \ldots, n$$

- La carga académica del periodo j debe ser menor o igual al máximo establecido de γ:

$$c_j \leq \gamma, \forall j = 1, \ldots, n$$

- La cantidad de cursos del periodo j debe ser mayor o igual que el mínimo permitido de δ:

$$\sum_{i=1}^{m} x_{ij} \geq \delta, \forall j = 1, \ldots, n$$

- La cantidad de cursos del periodo j debe ser menor o igual que el máximo permitido de ϵ:

$$\sum_{i=1}^{m} x_{ij} \leq \epsilon, \forall j = 1, \ldots, n$$

- Finalmente, las restricciones que indican la naturaleza de las variables c y c_j que son valores positivos mayores a 0. Y x_{ij}, la cual es de tipo binaria y puede tomar los valores 0 o 1.

$$\begin{aligned} c &\in \mathcal{N} \\ c_j &\in \mathcal{N}, \ \forall j \\ x_{ij} &\in \{0, 1\}, \ \forall i, j \end{aligned}$$

4.5. Espacio de Búsqueda

Se debe elegir en qué periodo estará cada curso, por lo que si se consideran p Periodos y c Cursos, el espacio de búsqueda es:

$$c^p$$

5. Representación

La solución entregada por el algoritmo es un vector de la clase *Period* con un tamaño igual a la cantidad de periodos que tiene la malla a utilizar, esta clase se compone de 3 elementos principales:

- nCourses: Indica la cantidad de cursos que tiene el periodo, necesario para cumplir restricciones de mínimo y máximo de cursos por periodo.
- nCredits: Indica la cantidad total de créditos que tiene el periodo, necesario para cumplir restricciones de mínimo y máximo de créditos por periodo.
- courses: Es un vector con los identificadores de cada curso que fue asignado al periodo.

Los identificadores de los cursos son procesados por una función para entregar la sigla del curso y la solución sea mostrada de manera más entendible.

6. Descripción del algoritmo

El programa comienza obteniendo los parámetros y datos a partir de un archivo de texto, con el cual instancia las clases *Course* que son necesarias para el algoritmo. Esta clase tiene como componentes relevantes:

- credits: La cantidad de créditos del curso.
- name: La sigla correpondiente al curso.
- pheromones: Un vector con el mismo largo que la cantidad de periodos y que indica la cantidad de feromonas para cada periodo.
- heuristic: Un vector con el mismo largo que la cantidad de periodos y que indica una tendencia a ser parte de ciertos periodos dependiendo de su cantidad de prerrequisitos.
- probs : Un vector con el mismo largo que la cantidad de periodos y que es una combinación del conocimiento heurístico y la cantidad de feromonas en cada periodo. Además, este vector también entrega la información de disponibilidad del curso en algún periodo dado, si la probabilidad es 0, significa que ese curso no puede ser asignado en ese periodo.
- prerreq: Un vector con las siglas de los cursos que deben ser asignados previamente.
- isPrerreqOf: Un vector con las siglas de los cursos que necesitan que el actual curso sea asignado previamente.

El resultado del proceso de obtención de parámetros y datos es un vector con todos los cursos que se deben asignar, para el cual se utiliza una estructura *map* para hacer una correspondencia entre los identificadores de cada curso con su sigla. El algoritmo utilizado se compone de 2 partes principales:

- Solución Inicial: Es utilizado el algoritmo de *Min-Max Ant System* para obtener una solución inicial que contenga la mayoría de los cursos asignados a algún periodo y que no rompa restricciones, especialmente la de los pre-requisitos.
- Búsqueda local: Se itera sobre la solución inicial utilizando el algoritmo de *Simulated Annealing*, el que tiene por objetivo balancear la carga académica entre los periodos, moviendo cursos entre ellos mientras se satisfacen las restricciones.

6.1. Solucion Inicial

6.1.1. Feromonas

Las feromonas en este algoritmo son utilizadas para dar más peso a la probabilidad de cada curso de ser asignado en algún periodo, funcionando de manera similar a una ruleta con probabilidades distintas en cada periodo.

6.1.2. Información heurística

Previamente a comenzar con las iteraciones, la información heurística es agregada, la que depende de la cantidad de prerrequisitos que tiene el curso menos la cantidad de cursos de los que se es prerrequisito:

- Si el resultado es negativo o cero, se agregan más probabilidades de ser asignado dentro de los primeros periodos.
- Si el resultado es 1, se refuerzan los periodos en el medio.
- Si el resultado es mayor que 1, hay más probabilidades de que el curso sea asignado en los últimos periodos.

6.1.3. Evaporación y Stagnation

Cada cierta cantidad de iteraciones, las feromonas en cada curso son evaporadas disminuyendo su valor hasta un mínimo de 1.

En cuanto a la *stagnation*, se asume que cada cierto número de iteraciones, que suele ser entre el doble o el triple que las necesarias para la evaporación, las hormigas están en un estado de *stagnation*, lo que reinica todas las feromonas a su valor inicial.

6.1.4. Iteraciones de las hormigas

Cada hormiga, al empezar a iterar, actualiza el vector de probabilidades de cada curso con la información heurística y las feromonas siguiendo la fórmula:

$$\tau_{ij}(t)^{\alpha} * \eta_{ij}^{\beta}$$

Siendo $\tau_{ij}(t)$ las feromonas del curso i para el periodo j en un tiempo t y η_{ij} la información heurística del curso i para el periodo j.

Luego itera sobre cada curso, buscando un periodo en el cual asignarlo en base a una variable aleatoria.

Si se elige un periodo, se revisa la cantidad de cursos y créditos de ese periodo para confirmar la factibilidad de la elección. Al realizar la asignación, se suman los créditos del curso al periodo y el contador de cursos sube en 1. Además, se baja la probabilidad a 0 de todos los cursos que tengan como prerrequisito el curso asignado para el mismo periodo y los anteriores, asegurando que no puedan ser asignados antes que su prerrequisito. De igual forma, la probabilidad de ser asignado en el mismo periodo o en uno posterior baja a 0 para los prerrequisitos del curso asignado, esto es necesario para el mejoramiento de la solución que se realiza después.

En el caso de que el curso no encuentre un periodo en el que ser asignado, todos los cursos de los que es prerrequisito bajan su probabilidad a 0 para todos los periodos, de manera que cuando se itere sobre ellos, tampoco sean asignados.

6.1.5. Actualización de feromonas y mejoramiento de la solución

Luego de que todas las hormigas hayan pasado, se elige la mejor y la peor solución, esto está determinado por el número de cursos que fueron asignados, es decir, la mejor hormiga es la que asignó más cursos y la peor es la que menos cursos asignó.

La mejor hormiga es mejorada realizando un *swap* de cursos del periodo con la mayor y menor cantidad de créditos.

Posteriormente, se agregan feromonas a los periodos en los que cada curso fue asignado en la mejor hormiga. En cambio, se reduce la cantidad de feromonas de la peor hormiga.

Por ejemplo, si el curso 3 fue asignado al periodo 2 en la mejor hormiga, τ_{32} aumenta, en cambio, si el mismo curso 3 fue asignado al periodo 7 en la peor hormiga, τ_{37} disminuye.

Finalmente, si la mejor hormiga asignó más cursos que la mejor solución obtenida hasta ese momento, la solución de esa hormiga pasa a ser la mejor solución, en caso contrario, se mantiene la solución anterior.

6.2. Búsqueda local

Se utiliza un algoritmo de *Simulated Annealing* con un movimiento de *swap* para reparar la solución inicial obtenida anteriormente. La temperatura disminuye cada vez que el movimiento es exitoso.

Para comenzar, todos los cursos son examinados para actualizar su información de probabilidad, que será utilizada para confirmar el cumplimiento de las restricciones de prerrequisitos.

Luego, se eligen aleatoriamente dos periodos distintos, en los que se realiza el *swap* entre un curso de cada periodo si es que cumple con las restricciones del problema. Si se realiza el movimiento, las disponibilidades de todos los cursos son actualizadas en relación a sus prerrequisitos.

Esta nueva solución es comparada con la mejor obtenida hasta el momento para ver cual optimiza más el problema. Siendo mejor la que tiene una menor desviación estándar entre la cantidad de créditos de cada periodo.

Esta búsqueda local se realiza por una cantidad fija de iteraciones.

7. Experimentos

Para los experimentos, se considerará el algoritmo de *MMAS* por su cantidad de parámetros, además de probar con distintas semillas para comprobar la estabilidad del algoritmo. A menos que sea especificado lo contrario, los parámetros a utilizar son:

- Semilla: 2
- Cantidad de hormigas: 15
- Valor de α y β: Ambos en 1.
- Iteraciones totales: 1000
- Iteraciones para evaporación: 100
- Iteraciones para stagnation: 200
- Iteraciones para mejoramiento: 10
- Máxima cantidad de feromonas: 10

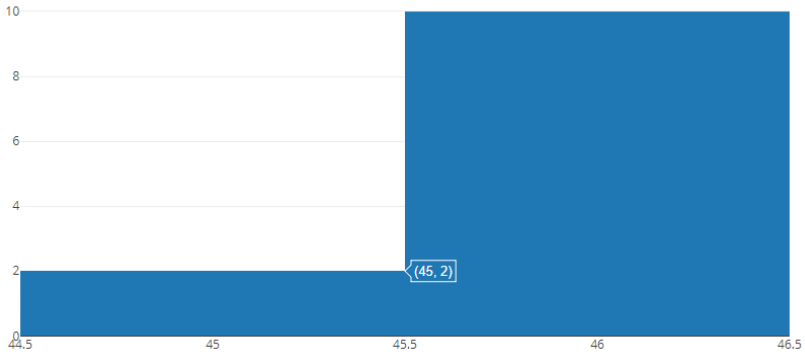
7.1. Semillas

Fueron probadas 12 semillas distintas para cada una de las 3 instancias.

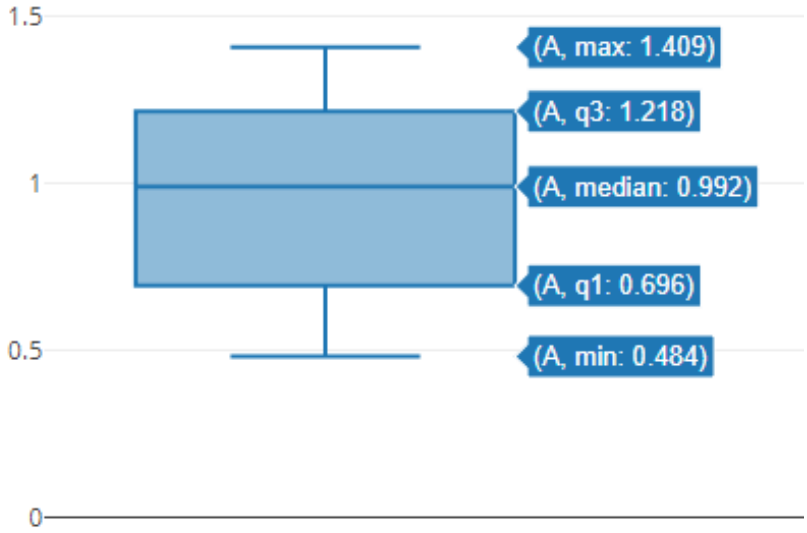
	bacp8		bacp10		bacp12	
Semillas	Cursos	StdDev	Cursos	StdDev	Cursos	StdDev
0	46	0.992	41	-	64	-
1	46	0.992	41	-	64	-
2	46	0.992	42	3.169	65	-
3	46	1.409	42	1.200	66	1.080
4	46	0.696	42	2.650	66	1.000
5	46	0.696	42	1.356	66	1.291
6	45	-	41	-	62	-
7	46	1.218	42	0.800	62	-
8	46	1.218	42	1.281	66	1.291
9	46	0.484	42	1.281	66	1.528
10	46	0.696	42	1.428	66	1.633
11	45	-	40	-	66	1.780

- bacp8

  El algoritmo asignó la totalidad de los cursos en 10 de las 12 ejecuciones, es decir, un 83 %.

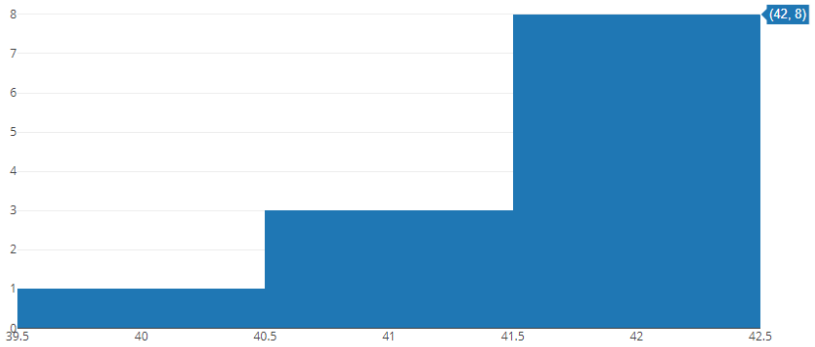


  Tomando en cuenta solo las ejecuciones en que todos los cursos fueron asignados, se obtiene el boxplot siguiente, en el que se muestra una distribución estable de los valores, sin outliers que puedan deberse a procesos aleatorios del algoritmo

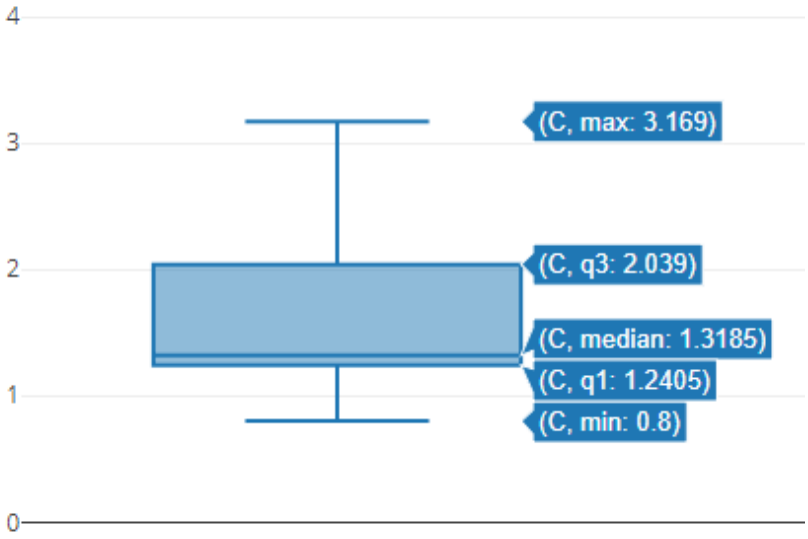


- bacp10

  El algoritmo asignó la totalidad de los cursos en 8 de las 12 ejecuciones, es decir, un 66 %.

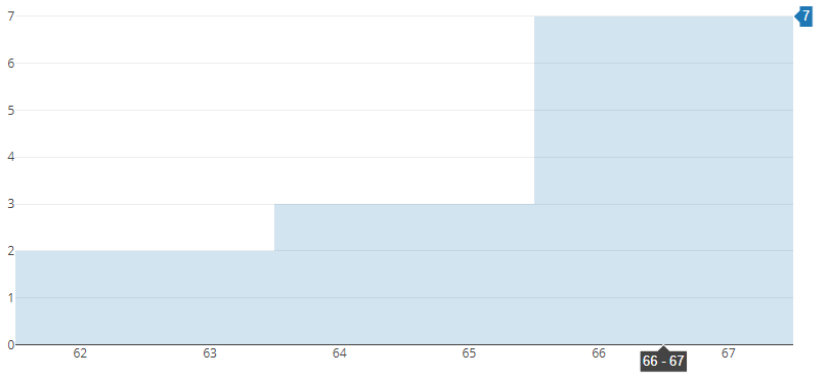


El boxplot fue realizado solo con las soluciones que asignaron la totalidad de los cursos. En este caso, hay una mayor variación en los datos.

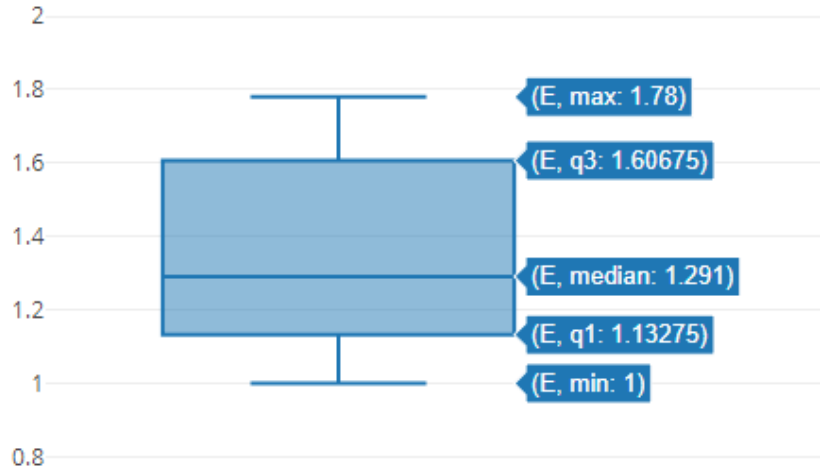


- bacp12

  El algoritmo asignó la totalidad de los cursos en 7 de las 12 ejecuciones, es decir, un 58 %.



El boxplot siguiente muestra que los datos se encuentran cercas unos a otros en comparación al anterior correspondiente a *bacp10*.



Dada las ejecuciones realizadas, el algoritmo podría considerarse robusto, ya que no presenta valores outliers y la diferencia entre los máximos y mínimos es menor a 1, con excepción de *bacp10* que presenta mayor variabilidad.

En cuanto a la cantidad de ejecuciones exitosas, en más de la mitad de las ocasiones se obtuvo una respuesta positiva, sin embargo, la consistencia de este resultado disminuye a medida que crecen las instancias. Debido a esto, es más difícil obtener resultados en instancias como *bacp12*.

7.2. Cantidad de hormigas

Instancia	Cantidad de hormigas	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	5	46	0.857	2.017
bacp8	10	46	0.992	3.426
bacp8	15	46	0.992	5.071
bacp8	30	46	0.696	9.163
bacp10	5	42	3.104	1.991
bacp10	10	42	1.200	3.218
bacp10	15	42	3.169	4.699
bacp10	30	42	1.114	8.384
bacp12	5	65	1.299	2.947
bacp12	10	66	2.198	6.014
bacp12	15	65	1.0274	7.444
bacp12	30	66	0.816	13.976

En las tres instancias se obtuvo un mejor desempeño al utilizar 30 hormigas por cada iteración, además se destaca que en *bacp12*, esta cantidad de hormigas consiguió asignar todos los cursos, a diferencia de la cantida de 5 y 15. La conclusión es que 30 hormiga es un número adecuado para el algoritmo.

7.3. Alfa α y Beta β

Solo es necesario variar uno de los parámetros, pues uno nivela al otro. Por lo que se harán modificaciones sobre el parámetro β.

Instancia	Beta β	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	0	46	0.857	4.968
bacp8	1	46	0.992	5.147
bacp8	5	46	0.992	5.564
bacp10	0	42	1.497	4.757
bacp10	1	42	3.167	4.787
bacp10	5	42	0.917	5.018
bacp12	0	62	2.532	8.132
bacp12	1	65	1.027	7.726
bacp12	5	66	1.414	8.174

Para el caso de *bacp8*, el valor de β no tiene un gran impacto en el resultado, pero en el de *bacp10* baja considerablemente la desviación estándar de 3,167 a 0,917. En el caso de *bacp12*, un β igual a 5 logra asignar todos los cursos de la instancia. En este caso, β indica la relevancia de la información heurística, mientras más grande sea, más peso tiene al calcular las probabilidades, por lo que se concluye que priorizar este aspecto entrega mejores resultados.

7.4. Iteraciones totales

Instancia	Iteraciones	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	100	46	1.218	0.941
bacp8	500	46	1.409	2.884
bacp8	1000	46	0.992	5.203
bacp10	100	42	3.169	0.723
bacp10	500	42	3.200	2.419
bacp10	1000	42	3.169	4.456
bacp12	100	65	0.943	1.288
bacp12	500	65	1.106	4.677
bacp12	1000	65	1.027	7.913

Se puede concluir que con una cantidad de iteraciones igual a 1000, se obtienen buenos resultados para las instancias *bacp8* y *bacp10*. En el caso de *bacp12*, ninguna combinación de iteración logró asignar todos los cursos, por lo que indica que los otros parámetros tienen mayor importancia a la hora de mejorar el algoritmo.

7.5. Iteraciones para evaporación

Estas pruebas fueron realizadas sin *stagnation*.

Instancia	Iteraciones	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	-	46	0.992	4.785
bacp8	100	46	0.992	5.199
bacp8	500	46	0.992	5.031
bacp10	-	42	3.169	4.204
bacp10	100	42	3.169	4.527
bacp10	500	42	3.169	4.461
bacp12	-	65	1.027	7.322
bacp12	100	65	1.027	7.804
bacp12	500	65	1.027	7.544

No hubo cambios significativos con las diferencias de iteraciones para la evaporación, lo que podría indicar que las feromonas llegan al máximo en tan pocas iteraciones que la evaporación no produce un efecto notorio.

7.6. Iteraciones para *stagnation*

Instancia	Iteraciones	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	-	46	0.992	4.976
bacp8	200	46	0.992	4.585
bacp8	500	46	0.992	5.255
bacp10	-	42	3.169	4.628
bacp10	200	42	3.169	4.539
bacp10	500	42	3.169	4.340
bacp12	-	65	1.027	7.212
bacp12	200	65	1.027	7.814
bacp12	500	65	1.027	7.561

No hubo cambios significativos con las diferencias de iteraciones para el estado d *stagnation*, lo que podría indicar que las feromonas llegan al máximo en tan pocas iteraciones que la *stagnation* no produce un efecto notorio.

7.7. Iteraciones de mejoramiento de solución

Instancia	Iteraciones	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	1	46	0.857	4.717
bacp8	10	46	0.992	5.259
bacp8	20	46	0.992	4.910
bacp10	1	41	2.236	4.506
bacp10	10	42	3.169	4.481
bacp10	20	42	3.169	4.361
bacp12	1	66	1.477	7.681
bacp12	10	65	1.027	7.603
bacp12	20	65	1.027	7.825

Este experimento no es concluyente, puesto que para la misma cantidad de iteraciones, se obtienen resultados diferentes en las instancias. Un ejemplo es que para una cantidad de 1, *bacp8* obtiene mejores resultados, pero *bacp10* no logra asignar todos los cursos, mientras que *bacp12* obtiene su mejor resultado con este valor.

7.8. Máxima cantidad de feromonas

Instancia	Cantidad máxima de feromonas	Cursos asignados	Desviación Estándar	Tiempo [s]
bacp8	10	46	0.992	4.758
bacp8	30	46	0.484	4.770
bacp8	100	46	0.484	5.416
bacp10	10	42	3.169	4.252
bacp10	30	42	2.200	4.445
bacp10	100	42	2.200	4.447
bacp12	10	65	1.027	7.799
bacp12	30	64	2.871	7.665
bacp12	100	64	1.605	7.387

Tomando en cuenta las dos primeras instancias, *bacp8* y *bacp10*, se concluye que la cantidad máxima de feromonas debe ser entre 30 y 100, puesto que el resultado es mejorado. En el caso de la tercera instancia *bacp12*, ninguna solución contiene los 66 cursos, por lo que otro parámetros deben ser variados para mejorar los resultados en esta instancia.

8. Resultados

El algoritmo utilizó los mejores parámetros que pudieron concluirse de los experimentos en la sección anterior, los que fueron fijados en:

- Semilla: 2
- Cantidad de hormigas: 30
- Valor de α y β: α en 1 y β en 2.
- Iteraciones totales: 1000
- Iteraciones para evaporación: 100
- Iteraciones para stagnation: 200
- Iteraciones para mejoramiento: 10
- Máxima cantidad de feromonas: 50

Además, dos instancias fueron agregadas correspondientes a las carreras de Ingeniería Civil Informática *bacpINF10*, hasta el décimo semestre, e Ingeniería Civil Telemática *bacpTEL12*, de 12 semestres. Ambas de la Universidad Técnica Federico Santa María.

- bacp8, con una desviación estándar de: 0,484

```
Period 1 (17): fis100-mat190-mat192-iwi131-iwn170
Period 2 (17): mat191-mat193-iei134-iei141-iei162
Period 3 (17): iwg101-fis101-iei133-iei231-iwn261
Period 4 (16): iei142-iei271-iei238-iwi365-iwn270-hcw310
Period 5 (16): mat194-iei272-iei161-hrw130-iei218-dew100
Period 6 (17): mat195-iei241-iei233-iei261-iei273
Period 7 (16): dew101-fis102-hxwxx1-iei132-iei232-iei262
Period 8 (17): dewxx0-hcw311-hxwxx2-iei281-hfw120-iei274-iei219-iei248
Cursos asignados: 46 de 46
Tiempo de ejecucion: 8.592 segundos
```

- bacp10, con una desviación estándar de: 1,114

```
Period 1 (13): dew100-qui010-iwi131-ili260-iwn261-
Period 2 (13): ili134-ili151-icdxx1-icdxx2-
Period 3 (13): fis100-mat021-dew101-hrwxx3-hrwxx1-
Period 4 (13): ili153-ili270-iwn170-iwg101-hrwxx2-
Period 5 (13): fis110-ili135-ili245-
Period 6 (13): mat022-dewxx0-ili253-ili263-
Period 7 (12): fis120-hcw310-mat023-iwn270-
Period 8 (15): mat024-ili239-ili243-ili362-
Period 9 (13): fis130-ili236-ili280-hcw311-
Period 10 (16): fis140-ici344-ili332-ili355-
Cursos asignados: 42 de 42
Tiempo de ejecucion: 8.36042 segundos
```

- bacp12, con una desviación estándar de: 0,913

```
Period 1 (16): mat111-mat121-iwi131-ili260-hxwxx1-hcw310
Period 2 (16): mat112-mat122-ili134-ili273-dew100
Period 3 (17): fis100-mat123-ili231-ili243-ili363
Period 4 (17): mat113-ili151-ild208-ili274-ici315-hrw150
Period 5 (19): dew101-ili142-ili252-iwn270-ici344-qui104
Period 6 (17): iwm185-iwn170-ili242-iwn261-ili362-ici367
Period 7 (18): mat124-ili238-ili275-ili237-iln230
Period 8 (16): fis110-dewxx0-mat270-hrw110-iwi365-hcw311
```

```
Period 9 (16): fis120-fis130-mat210-ili334
Period 10 (17): iwg101-fis140-hxwxx2-mat260-hrw100-ili331
Period 11 (17): ili221-ili281-ici393-ici382-ici313
Period 12 (18): ile260-ili355-ili381-ici332-icn336-ici314
Cursos asignados: 66 de 66
Tiempo de ejecucion: 14.3651 segundos
```

- bacpINF10, con una desviación estándar de: 0,922

```
Period 1 (16): iwi131-mat021-hrw132-qui010-inf301
Period 2 (19): fis100-mat022-iwg101-inf134-inf152-inf312
Period 3 (16): dew100-fis110-inf239-inf4-inf311-inf313
Period 4 (17): mat023-inf260-inf253-inf6-inf303-inf304
Period 5 (16): mat024-inf155-inf245-inf270-inf302
Period 6 (17): inf1-fis120-inf2-inf236-inf276-inf292-hrw133
Period 7 (16): dew101-fis130-iwn170-inf246-inf225-inf5-inf7
Period 8 (16): fis140-inf280-inf322-inf266-inf314
Period 9 (16): inf3-inf221-inf256-icn270-inf293-inf360
Period 10 (16): inf285-inf343-inf295-inf228-inf309
Cursos asignados: 58 de 58
Tiempo de ejecucion: 11.4191 segundos
```

- bacpTEL12, con una desviación estándar de: 1,027

```
Period 1 (18): dew100-hcw100-iwg399-fis100-qui010-iwg397-tel360
Period 2 (17): mat021-tel101-iwg101-hcw101-iwg394
Period 3 (16): mat022-dew101-iwg396-hfw144-hrw102-tel012
Period 4 (16): tel102-elo322-elo204-hcw102-tel021-dew110
Period 5 (19): mat023-elo329-iln250-tel011-iwn261-tel013
Period 6 (17): elo320-tel343-iwg398-hrw101-tel132-tel241
Period 7 (16): hcw200-tel315-iwn170-iwg395-tel317
Period 8 (18): fis110-inf239-mat024-hcw201-tel341-elo321
Period 9 (17): fis120-fis130-mat270-tel342-tel335
Period 10 (17): tel131-tel222-inf236-tel312-tel211-hcw202
Period 11 (18): tel231-elo241-elo212-tel354-elo328-tel329
Period 12 (19): inf225-elo211-fis140-elo341-tel252-tel236
Cursos asignados: 70 de 70
Tiempo de ejecucion: 14.0658 segundos
```

Dentro de la literatura, enontramos otro paper que utiliza algoritmos de hormigas, publicado por Rubio et. al [8] en el año 2013. Una tabla con comparaciones de la desviación estándar obtenida por los resultados es presentada a continuación:

Instancia	Rubio et. al [8]	MMAS con SA
bacp8	0.48	0.484
bacp10	0.42	1.114
bacp12	0.69	0.913

En el caso de *bacp8*, los resultados son similares, pero en las otras instancias, *bacp10* y *bacp12*, los resultados son peores. Esto indicaría que se requiere una mejor implementación en el algoritmo reparador.

9. Conclusiones

El algoritmo de colonia de hormigas es utilizado para obtener la solución inicial que contenga todos los cursos asignados, para este propósito se comportó de buena manera, obteniendo resultados positivos en más del 50 % de las ejecuciones realizadas para todas las instancias originales. Debido a los prerrequisitos necesarios, obtener una solución inicial que satisfaga todas las restricciones no es una tarea trivial, por lo que se destaca que en las 5 instancias utilizadas, el algoritmo asignó todos los cursos en todas ellas, permitiendo que el algoritmo reparador encontrara soluciones muy cercanas al óptimo, siendo todas las desviaciones estándar cercanas a 1 o inferior.

Respecto a los experimentos, los resultados obtenidos utilizando distintas semillas indica que en la mayoría de las ejecuciones debiesen encontrarse soluciones completas, es decir, con la totalidad de los cursos asignados, por lo que el algoritmo debiese ser útil al momento de usarse. En cuanto a los parámetros, las mayores diferencias se vieron con la cantidad de hormigas, el valor de β y el máximo de feromonas, mientras que la cantidad de iteraciones, ya sean totales, de evaporación o *stagnation*, no afectan en gran medida a los resultados del algoritmo.

Dentro de algunas mallas curriculares actuales existen cursos que tienen como prerrequisito haber aprobado una cierta cantidad de créditos, restricción que este algoritmo no toma en cuenta, por lo que es un aspecto de mejora a desarrollarse en un trabajo a futuro. Además, el algoritmo reparador implementado se encuentra dentro de los más simples, por lo que otra forma de mejorar los resultados es utilizar otro tipos de algoritmos que puedan explorar más y encontrar soluciones más óptimas.

Finalmente, respecto al tiempo de ejecución, tomando en cuenta que las instancias son tomadas de casos reales que fueron ligeramente simplificados, una ejecución menor a 1 minuto es bastante aceptable, considerando también que una variación de los parámetros puede entregar desde mejores soluciones a otras iguales pero con una distribución distinta de los cursos.

Referencias

[1] Carlos Castro and Sebastian Manzano. Variable and value ordering when solving balanced academic curriculum problems. *arXiv preprint cs/0110007*, 2001.

[2] Marco Chiarandini, Luca Di Gaspero, Stefano Gualandi, and Andrea Schaerf. The balanced academic curriculum problem revisited. *Journal of Heuristics*, 18(1):119–148, 2012.

[3] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. CSPLib problem 030: Balanced academic curriculum problem (bacp). `http://www.csplib.org/Problems/prob030`.

[4] Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Modelling a balanced academic curriculum problem. In *Proceedings of CP-AI-OR-2002*. Citeseer, 2002.

[5] Tony Lambert, Carlos Castro, Eric Monfroy, and Frédéric Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. In *International Conference on Artificial Intelligence and Soft Computing*, pages 410–419. Springer, 2006.

[6] Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville, Pierre Dupont, et al. A cp approach to the balanced academic curriculum problem. In *Seventh International Workshop on Symmetry and Constraint Satisfaction Problems*, volume 7, 2007.

[7] Lorna V Rosas-Téllez, José L Martínez-Flores, and Vittorio Zanella-Palacios. Solution to the balanced academic curriculum problem using tabu search. *Computer Technology and Application*, 3(9), 2012.

[8] José-Miguel Rubio, Wenceslao Palma, Nibaldo Rodriguez, Ricardo Soto, Broderick Crawford, Fernando Paredes, and Guillermo Cabrera. Solving the balanced academic curriculum problem using the aco metaheuristic. *Mathematical Problems in Engineering*, 2013, 2013.

[9] Kadri Sylejmani, Arbnor Halili, and Arbnor Rexhepi. Balancing academic curricula by using a mutation-only genetic algorithm. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2017 40th International Convention on*, pages 1189–1194. IEEE, 2017.

[10] Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and J-C Régin. Constraint programming in opl. In *International Conference on Principles and Practice of Declarative Programming*, pages 98–116. Springer, 1999.