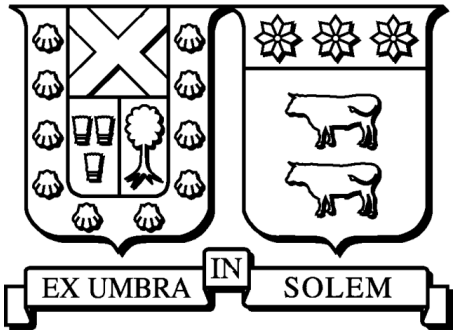
Universidad Técnica Federico Santa María
Departamento de InformáticaValparaíso
Valparaíso - Chile



Una metaheurística para la resolución de Balanced Academic Curriculum Problem [BACP]

Darío Andrés Canales Rojas
dario.canales@alumnos.usm.cl

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Dra. María Cristina Riff Rojas
Profesor Correferente: Dr. Hubert Hoffmann Nagel
Agosto 2014

Agradecimientos

Agradezco a mis padres por, con mucho esfuerzo, darme una buena educación, les agradezco además por motivarme a aprender desde niño y por apoyarme en mis fracasos y celebrar mis triunfos, sin ustedes no habría llegado tan lejos.

Agradezco a la Javi por ser una excelente persona, acompañándome, entendiéndome y ayudándome siempre. Con ella he superado muchas adversidades de la vida y espero que sigamos mejorando como personas.

Agradezco a la profesora María Cristina Riff por creer en mí y darme su gran apoyo, por mostrarme el camino por delante, enseñarme herramientas y darme más motivación para seguirlo. A la profesora Elizabeth Montero porque sin conocernos demasiado me ha brindado de toda su amabilidad y carisma.

A todos aquellos que de una u otra forma me ayudaron a llegar hasta aquí.

Al azar que me permitió existir y disfrutar tanto del aprender.

Resumen

El Balanced Academic Curriculum Problem, BACP de ahora en adelante, consiste en la asignación de un conjunto de cursos a los periodos de duración de una carrera universitaria, cumpliendo con las restricciones asociadas a la carga académica de cada curso y cantidad de cursos por periodo. El objetivo del problema es lograr una asignación balanceada, que corresponde a aquella que logre distribuir los cursos más equitativamente entre los distintos periodos. En esta memoria se presenta un estudio de los distintos enfoques que existen en la literatura que buscan resolver el problema. Posteriormente se propone una metaheurística basada en Tabu Search que resuelve las principales instancias propuestas en la literatura con alta eficacia y eficientemente, junto con una herramienta de visualización de datos, que ayuda a analizar los comportamiento de metaheurísticas basadas en BACP.

Abstract

The Balanced Academic Curriculum Problem, BACP hereafter, consists of assigning a set of courses to periods of a university major, satisfying the constraints associated to the academic load of each course and the amount of courses per period. The objective of this problem is to achieve a balanced assignment, which is the one that achieves the most equitable distribution of courses in the different periods. In this thesis a study of the different approaches that exists in the literature that seeks to solve the problem is presented. Subsequently, a metaheuristic based on Tabu Search is proposed, which solves the main instances proposed in the literature with high efficacy and effectiveness, along with a data visualization tool that helps to analyze the behavior of the BACP based metaheuristics.

Índice general

1. Introducción **1**

2. Definición del problema **3**
2.1. Objetivos y variantes del problema 4
2.2. Complejidad del problema 6
2.3. Resumen 6

3. Estado del Arte **7**
3.1. Instancias y benchmark 12
3.2. Modelo Matemático 12
3.2.1. Modelo de programación entera (IP) 12
3.3. Resumen 15

4. Descripción del algoritmo **17**
4.1. Tabu Search 17
4.1.1. Representación 18
4.1.2. Generación de la solución inicial 18
4.2. Tabu Search para BACP 25
4.2.1. Movimiento 25
4.2.2. Lista Tabu 27
4.2.3. Función objetivo, función de evaluación y criterio de selección . 28
4.2.4. Algoritmo propuesto 30
4.3. Conclusiones 30

5. Resultados Experimentales **32**
5.1. Especificaciones del equipo de trabajo 32
5.2. Métodos de evaluación 32
5.3. Experimentos 32
5.4. Resultados 34
5.4.1. Instancias CSP Lib 34
5.4.2. 100 instancias obtenidas de Bacp12 38
5.5. Conclusiones 42

6. Herramienta de visualización para metaheurísticas que solucionen BACP 43
6.1. Motivación 43
6.2. Herramienta propuesta 45
6.2.1. Opciones de visualización de malla curricular 45
6.2.2. Secciones de la visualización 46
6.3. Limitaciones de la herramienta 47
6.4. Conclusiones 48

7. Conclusiones 49

Bibliografía 51

1 Introducción

En la actualidad es muy frecuente encontrar la aplicación de técnicas provenientes de la inteligencia artificial e investigación de operaciones a problemas relacionados con asignación y scheduling, en donde dado el gran espacio de búsqueda, se requiere el uso de técnicas eficientes y eficaces que generalmente están asociadas a búsquedas basadas en heurísticas.

Esta memoria se centra en diseñar el plan de estudio de licenciaturas o carreras universitarias, donde se busca asignar un conjunto de cursos a los distintos periodos de duración de una carrera. Uno de los factores más importantes a considerar corresponde a la carga académica que cada curso conlleva para el estudiante. Esta carga se expresa frecuentemente usando el concepto de "crédito" como unidad de medida, que dimensiona la cantidad de dedicación requerida para un curso específico. De esta manera, el problema consiste en distribuir el conjunto de cursos balanceadamente, es decir, que la carga académica total dada por la suma de los créditos de cada curso tenga una distribución homogénea, de modo que la carga sea lo más uniforme posible a lo largo de toda la carrera. Por otro lado, se debe considerar que asignar varios cursos que representen demasiada carga en un mismo periodo podría perjudicar el rendimiento de los alumnos, y que asignar muy pocos cursos en ciertos periodos podría causar un incremento en la duración de la carrera.

Además de lo anterior, existen una serie de distintos factores a tomar en consideración que incrementan la complejidad del problema tales como restricciones de precedencia entre cursos. Se tiene además la restricción de no asignar menos de un mínimo número de cursos en cada período académico, que un curso no pueda ser asignado sin antes haber aprobado otro (restricción de prerrequisitos), asignar todos los cursos dentro de la duración de la carrera, entre otros. Además, es preciso notar que entre mayor sea la duración de la carrera más complejo será el proceso de planificación, ya que se incrementa el número de cursos y periodos y, consecuentemente, el número de maneras distintas en que éstos pueden ser asignados. Esto hace que el problema de asignación balanceada de mallas curriculares académicas sea fuertemente NP-completo, tal como se señala en [2].

Este trabajo está organizado de la siguiente manera. En el Capítulo 2 se estudiará el problema en profundidad, indicando cada una de los componentes y restricciones que se deben considerar. Posteriormente en el Capítulo 3 se revisarán algunas de las técnicas utilizadas para atacar el problema, así como también los distintos enfoques

que se le ha dado en la literatura, junto con los diferentes resultados obtenidos a partir de la aplicación de las técnicas propuestas y de tendencias actuales dando paso a la presentación formal de algunos modelos matemáticos que sirven para abordar el problema. En el Capítulo 4 se propone un algoritmo para solucionar el problema Balanced Academic Curriculum Problem (BACP) mediante la metaheurística de búsqueda Tabu Search. En el capítulo 5 se presentan los resultados experimentales, el análisis de estos y las comparaciones con los mejores resultados de la literatura. En el Capítulo 6 se presenta una herramienta de visualización que permite generar representaciones gráficas del BACP. Finalmente en el Capítulo 7 se presentan las conclusiones generales y trabajo futuro.

2 Definición del problema

El problema de diseño de currículum balanceado (BACP) trata sobre la asignación de un conjunto de cursos al conjunto de periodos de duración de una carrera. Como no todos los cursos poseen la misma dificultad, es necesario considerar que una buena asignación es aquella que distribuye los cursos agrupándolos, de tal manera que en cada periodo la dificultad esté uniformemente distribuída o, al menos, que ésta sea lo más similar posible entre los diferentes períodos. Para medir de alguna manera los resultados, comúnmente se asigna un número específico de créditos para cada curso, representando así la cantidad de esfuerzo en términos de tiempo de dedicación que requiere un alumno para ese curso. Además de lo anterior, algunos cursos cuentan con relaciones de precedencia o prerrequisitos, es decir, no pueden ser asignados sin que antes el alumno haya cursado o aprobado un curso precedente.

Este problema, fue presentado en [1] y está listado como el problema número 30 de CSPLib [1], en donde además se propone un framework en base a las cuales se estructura:

- Currículum académico: se define como el conjunto de cursos totales pertenecientes al grado universitario que deben ser asignados. Además, pueden poseer relaciones de prerrequisitos, es decir, es necesario aprobar anteriormente un conjunto de cursos que habilitarán la inscripción del curso que tiene las precedencias.
- Número de períodos: la duración total de la carrera se divide en años académicos, los cuales a su vez se subdividen en periodos académicos, que son porciones de tiempo en la cual cada curso puede inscribirse. En base a esto, se impone la restricción de asignar todos los cursos dentro del total de número de periodos académicos de la carrera.
- Carga académica: cantidad de créditos asignados a un curso que señalan el esfuerzo asociado a éste en terminos de dedicación.
- Carga académica mínima y máxima de un período: cantidad de carga mínima y máxima total que un periodo académico puede contener expresada en créditos, con el fin de cumplir con un avance mínimo requerido por la institución (carga mínima) y no sobrecargar periodos con un nivel de exigencia excesiva que pueda perjudicar el rendimiento de los alumnos (carga máxima).

[1] http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob030/index.html

- Mínimo y máximo número de cursos de un período: cantidad de cursos mínimos y máximos que un periodo académico puede contener, con el fin de cumplir con los mismos objetivos que en el punto anterior.

Un ejemplo de malla curricular puede visualizarse en la Figura 2.1:

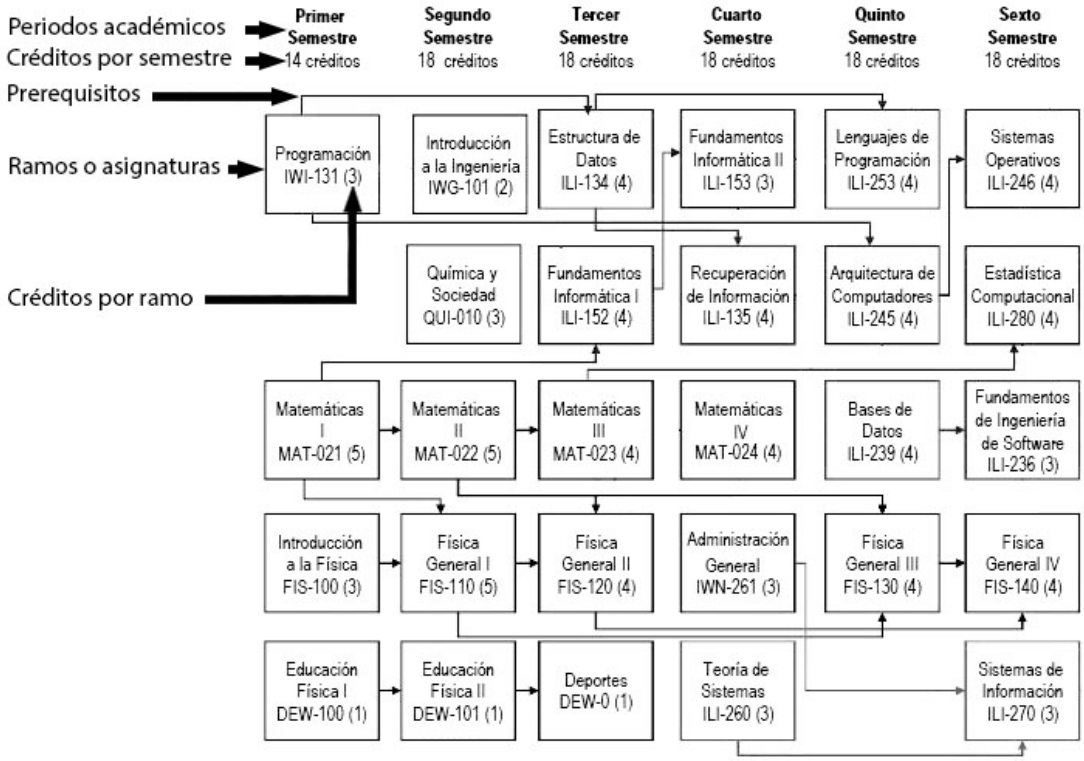


Figura 2.1: Representación gráfica de una malla curricular. Elaboración propia a partir de malla de Ingeniería Civil Informática Universidad Técnica Federico Santa María.

El proceso de búsqueda de soluciones que satisfagan las restricciones consiste esencialmente en encontrar asignaciones para todos los cursos en cada uno de los periodos de duración de la carrera, de tal manera que se cumpla con las restricciones de máximo y mínimo de créditos y cursos.

2.1. Objetivos y variantes del problema

El problema BACP puede verse como un problema de satisfacción de restricciones, en donde sólo se busca encontrar soluciones factibles o como un problema de minimización, al considerar que una solución será mejor en la medida en que la asignación encontrada

disminuya el máximo de las cargas académicas asignadas a todos los periodos (cargas balanceadas) [1, 2, 5]. Por otro lado, también es posible señalar que si bien la idea de balance en la malla curricular puede ser interpretada como la señalada anteriormente, también pueden definirse diversas nociones cuantitativas que dan origen a diferentes funciones objetivo, entre las cuales se destacan las siguientes variantes (enumerados en [15]):

- Rango de cargas académicas[14]: donde se busca minimizar la diferencia entre la carga académica máxima y la carga académica mínima de todos los periodos.
- Máxima diferencia con respecto a la media[10]: donde se busca minimizar la máxima diferencia entre las distintas cargas académicas y la carga académica promedio
- Diferencia absoluta con respecto a la media[15]: donde se busca minimizar la suma de las diferencias absolutas de las cargas académicas respecto de la carga académica promedio

Por otro lado en [13] se incorpora una modificación que añade una restricción para especificar que ciertos cursos están fijas en un periodo, además se propone agregar al problema que la carga académica máxima y mínima, así como la cantidad de cursos máxima y mínima pueda variar dependiendo del periodo. Esto incorpora mayor flexibilidad al modelo del problema, a la vez que se mantiene la dificultad para resolverlo.

Además, en el año 2009 en [2]se proponen nuevas aproximaciones al problema dando origen a dos nuevas características, las cuales son:

- Currícula: Representa un conjunto de curriculums, esto añade la idea de que alumnos también tienen un cierto grado de decisión incorporando opciones personales y que pueden escoger entre distintas opciones a inscribir en un periodo o período específico. Siendo un curriculum un posible conjunto de cursos representando una posible selección completa de un estudiante, se extiende la idea a considerar varias currículas. Diferentes currículas pueden compartir algunos cursos.
- Preferencias de profesores: que añade la idea de que los profesores pueden tener preferencias por enseñar sólo en periodos académicos específicos. De esta manera se tendrá un conjunto de asignaciones indeseables de profesores, restringiendo las asignaciones de ese curso sólo para aquellos periodos de tiempo deseables.

La incorporación de estas nuevas características añade el concepto de problema balance de curriculum generalizado o GBACP, por sus siglas en inglés. La Figura 2.2 muestra una pequeña instancia de GBACP con 6 cursos, dos años (cada uno con 2 periodos académicos o periodos cada uno), 3 curriculas y 4 pares de preferencias descritas con la forma $(asignatura, periodo)$ lo cual describe que ese curso no será dictada en ese período, producto de las preferencias de un profesor en particular.

$$C := \{ \quad A \quad B \quad C \quad D \quad E \quad F \quad \}$$
$$r := \{ \quad 5 \quad 10 \quad 5 \quad 4 \quad 20 \quad 7 \quad \}$$
$$P := \{ \quad 1 \quad 2 \quad 3 \quad 4 \quad \}$$
$$\mathcal{Q} := \{[A, B, C, D], [B, C, D, F], [A, B, E, F]\}$$
$$U := \{(B, 1), (B, 3), (A, 2), (A, 4)\}$$

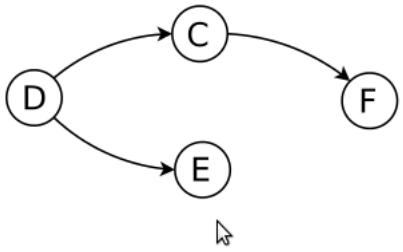


Figura 2.2: Instancia GBACP, C representa el conjunto de cursos a asignar, r es el conjunto de créditos correspondientes a cada curso, P representa el número de periodos académicos, Q representa el conjunto de curriculas y U el conjunto de preferencias de profesores[2].

2.2. Complejidad del problema

Tanto BACP como GBACP son problemas de complejidad fuertemente NP-completo [2], lo cual significa que en la práctica, para cualquier función objetivo computable en tiempo polinomial se necesitará un tiempo computacional y el número de operaciones o cálculos necesarios para resolver este tipo de problemas que crece exponencialmente a medida que crece el tamaño de la instancia a resolver.

2.3. Resumen

Este capítulo permite comprender en qué consiste el problema Balanced Academic Curriculum Problem, sus componentes, objetivos y principales variantes. En el siguiente capítulo se hará una revisión de avances y técnicas que se han utilizado en la literatura, así como también el modelo matemático y las restricciones asociadas al problema.

3 Estado del Arte

La mayor parte de los autores recientes en la literatura se refieren a [1] como uno de los trabajos pioneros en la formulación de un modelo para BACP, tanto así que el primer modelo propuesto es referido comúnmente como el modelo propuesto por «Castro and Manzano». Además de esto, una aproximación anterior a este trabajo fue realizada en [16] donde se aplicó programación entera para resolver un problema de 6 periodos académicos.

En la actualidad, la cantidad de publicaciones sobre el tema se ha diversificado bastante, proponiéndose varios modelos que abordan el problema desde dos perspectivas principales, la programación entera y la satisfacción de restricciones. Al respecto cabe destacar que como BACP es un problema fuertemente NP-Completo [2], para instancias reales se hace inviable la utilización de métodos completos en su resolución. Debido a eso se recurrió a heurísticas y métodos incompletos que presentan soluciones que, si bien pueden no ser óptimas, son factibles y pueden ser obtenidas en un tiempo razonable.

En [1] se presenta el problema junto con dos representaciones, una basada en programación lineal entera y otra en satisfacción de restricciones. La idea principal del modelo fue generar la malla, estableciendo qué curso era asignado a cada período mientras se minimizaba la máxima carga académica de todos los periodos. Al respecto, se utiliza lp_solve [1] para resolver el modelo de programación lineal para carreras con 8, 10 y 12 periodos académicos. Para la instancia de 8 pudo encontrarse el óptimo en 1460 segundos, mientras que para la de 10 y 12 periodos no pudo encontrarse el óptimo. Para la última instancia ni siquiera se obtuvieron respuestas luego de 1 día de ejecución del algoritmo. Seguidamente, se utilizó el lenguaje de programación Oz para representar el modelo basado en satisfacción de restricciones, obteniéndose como resultado que fue posible encontrar soluciones para las instancias de 8 y 10 periodos antes de las 5 primeras horas de ejecución.

Después del trabajo desarrollado por Castro y Manzano, en [5] se propone la integración de distintos modelos para el problema BACP. Esta integración incluye los modelos de programación entera y satisfacción de restricciones (ILP y CP_1) antes mencionados, y un modelo de satisfacción de restricciones adicional CP_2, donde se logra aprovechar

[1] software para resolver modelos de programación lineal entera, disponible en: ftp://ftp.ics.ele.true.nl/pub/lp_solve/

la propagación de restricciones para obtener mejores rendimientos. A partir de estos modelos, se generaron los modelos híbridos $ILP + CP_2$ y $CP_1 + CP_2$. En definitiva, se hace hincapié en las dificultades que tenían los modelos ILP y CP_1 al manejar las restricciones de prerrequisitos, siendo este su principal problema, por lo que la idea central de la combinación de los modelos se basó en aprovechar las diferentes ventajas de cada modelo y beneficiarse especialmente de la propagación de restricciones. Los resultados obtenidos en la búsqueda de soluciones óptimas para los cinco modelos se muestran en la Figura 3.1:

Modelo	8 periodos, 46 cursos	10 periodos, 42 cursos	12 periodos, 66 cursos
ILP	3.45 $[s]$	4.23$[s]$	131.30$[s]$
CP_1	58.52$[s]$	-	-
CP_2	45.10$[s]$	-	-
$ILP + CP_2$	0.81$[s]$	8.44$[s]$	3.05$[s]$
$CP_1 + CP_2$	0.29$[s]$	0.59$[s]$	1.09$[s]$

Figura 3.1: Búsqueda de una solución óptima [5]. Tiempos en segundos para alcanzar el óptimo en las instancias BACP8, BACP10 y BACP12.

En particular, se destaca que para 10 y 12 periodos académicos tanto CP_1 como CP_2 no pudieron encontrar el óptimo, mientras que los modelos híbridos $CP_1 + CP_2$ y $ILP + CP_2$, a pesar de poseer una mayor cantidad de restricciones que los anteriores, pueden encontrar la solución óptima (debido a la propagación de restricciones y poda de los respectivos árboles de soluciones posibles).

En [6] se destaca la aplicación de un algoritmo genético (GA) con propagación de restricciones a un modelo de BACP basado en la satisfacción de restricciones. GA optimiza las soluciones en un espacio de búsqueda que va siendo progresivamente más pequeño gracias a la propagación de restricciones. Para lo anterior, se define un framework para la utilización de propagación de restricciones combinadas con el algoritmo evolutivo en base a la computación de un conjunto de funciones. En el caso de BACP, se definieron tres tipos de funciones específicas: funciones de reducción del dominio, funciones de división de dominio y funciones para la generación de la población. Los resultados obtenidos para la instancias de 8, 10 y 12 periodos utilizadas anteriormente con este modelo, comparados con los resultados de Castro y Manzano [1] se muestran en la Figura 3.2:

	8 periodos, 46 cursos	10 periodos, 42 cursos	12 periodos, 66 cursos
Castro y Manzano	1459.73 [s], 17 créditos (óptimo)	4.23[s],24 créditos (no óptimo)	-
GA + CP	15.05 [s], 17 créditos (óptimo)	34.84 [s], 14 créditos (óptimo)	35.20 [s], 18 créditos (no óptimo)

Figura 3.2: Comparación entre resultados Castro y Manzano con GA+CP de [6]. *Fuente: Elaboración propia.*

Se aprecia que utilizando el modelo propuesto en [6] se encontró rápidamente el óptimo en instancias de 8 y 10 periodos. Finalmente se analizan los resultados obtenidos, concluyendo que al probar cada una de las técnicas GA y CP por separado, la propagación de restricciones no logra encontrar una solución factible luego de 10 minutos de ejecución, mientras que GA es capaz de encontrar un óptimo, pero es 10 veces más lento que la hibridación entre GA y propagación de restricciones. Además de lo anterior, se propone un framework genérico de modelamiento para utilizar precisamente la resolución híbrida de BACP, el cual permite diseñar y manejar nuevas estrategias de este tipo de manera sencilla.

En otro aspecto de la investigación de BACP en diferentes publicaciones se proponen varias funciones objetivo, que representan distintas nociones de balance. En [15] se presenta una compilación de cuatro distintos objetivos junto con un modelo de programación lineal entera, en base al cual se realizó el diseño curricular de cinco carreras universitarias relacionadas con el área de ingeniería: Ingeniería en Agronomía (Iag), Ingeniería Civil (Ici), Ingeniería en Manufactura y Autopartes (Ima), Ingeniería Industrial (Ind) e Ingeniería Química (Iqm). Todos estos planes de carrera tienen una duración de 9 periodos académicos.

La tabla de la Figura 3.3 muestra información con las características detalladas de cada plan de estudios.

Código	Total de cursos	Total de créditos	Total de relaciones de prerrequisito
Lag	62	361	19
Ici	61	488	48
Ima	61	395	48
Ind	61	376	47
Lqm	60	402	53

Figura 3.3: Características de los planes de carrera [15].

Seguidamente, en la tabla de la Figura 3.4 se muestran las dimensiones de los problemas generados mediante los distintos modelos, considerando las carreras señaladas como posibles instancias a solucionar, dimensionando cada problema según el número de variables y el número de ecuaciones de los modelos generados. BACP-1 representa la función objetivo bajo el criterio de minimizar la máxima carga académica, BACP-2 representa la función objetivo basada en minimizar la distancia entre la carga académica máxima y mínima, BACP-3 representa el criterio de minimización de la máxima diferencia entre las cargas académicas y la carga promedio y BACP-4 el criterio de minimización de la suma de las diferencias absolutas entre las cargas académicas y la carga promedio.

Número de variables de los modelos BACP (variables enteras, total de variables)				
Código	BACP1	BACP2	BACP3	BACP4
Lag	558,550	558,560	558,577	558,576
Ici	549,550	549,551	549,568	549,567
Ima	549,550	549,551	549,568	549,567
Ind	549,550	549,551	549,568	549,567
Lqm	540,541	540,542	540,559	540,558
Número ecuaciones (Función objetivo + restricciones)				
Código	BACP1	BACP2	BACP3	BACP4
Lag	139	148	157	139
Ici	170	179	188	170
Ima	178	187	196	178
Ind	175	184	193	175
Lqm	174	183	192	174

Figura 3.4: Dimensiones de los problemas de BACP para las carreras señaladas en [15].Inicialmente se muestran las dimensiones de los problemas en función de su cantidad de variables enteras y cantidad de total de variables. Luego se muestran las dimensiones en función de la cantidad de restricciones.

A continuación se procedió a solucionar cada una de las instancias con los modelos, que se generaron con diferentes funciones objetivo, mediante la herramienta de resolución de modelos de programación entera HyperLingo [2]. La tabla de la Figura 3.5 muestra los resultados obtenidos por los cuatro modelos (donde * significa que no se alcanzó el óptimo).

Tiempo promedio al óptimo (segundos)				
Código	BACP1	BACP2	BACP3	BACP4
Lag	4,1	28,0	**	22,9
Ici	8,6	486,5	**	28,6
Ima	3,3	4,1	0,3	3,7
Ind	**	**	0,9	6,3
Lqm	0,6	177,1	0,8	6,9

Figura 3.5: Tiempo promedio empleado por diferentes modelos utilizando el software Hyperlingo, para llegar al óptimo [15].

Se aprecia que se destacaron principalmente los modelos basados en BACP-1 y BACP-4 por sobre BACP-2 y BACP-3, mostrando un buen desempeño y una buena calidad de sus soluciones, presentándose BACP-4 como la mejor opción para ese conjunto de planes de estudio. En base a esto, se destaca principalmente la importancia que tiene la influencia de la elección de la función objetivo en el modelo generado y, en consecuencia, su importancia en el tiempo de ejecución y calidad de las soluciones encontradas.

Junto con los criterios de balance señalados anteriormente, en [8] se evalúan además como criterios de balance la minimización de la desviación media $C(1)$, la minimización de la raíz de la desviación media $C(2)$ y la minimización de la máxima desviación de la media $C(3)$, todas respecto de la carga académica. Luego de varios experimentos con diferentes instancias, el criterio $C(1)$ se muestra como una buena alternativa ante el criterio común propuesto por Castro y Manzano. Además de esto, se definen nuevas reglas llamadas "reglas de dominancia", para así guiar la búsqueda de mejor manera basándose en la carga académica conjunto, de cursos junto a sus prerrequisitos, considerando además los prerrequisitos de sus prerrequisitos. Tales estrategias de dominancia fueron utilizadas luego para resolver las instancias mediante métodos de Branch-And-Bound y métodos basados en búsquedas locales, combinando Tabu Search con Branch-And-Bound. Además de buscar un óptimo, en este estudio se buscaba satisfacer las restricciones del problema realizando movimientos de la variable con el valor que mejore en mayor medida el valor de la función objetivo. Por último, se crea un generador de instancias para BACP con el fin de obtener una mayor variedad experimental y poder comparar algoritmos en instancias más grandes.

[2]Lindo Systems Inc., http://www.lindo.com

Por otro lado, en [2, 3] se propone una generalización del problema, en donde se cubren parámetros no cubiertos anteriormente, como son la opción de un estudiante de elegir los cursos que inscribe en un periodo académico específico y la opción de que los profesores puedan elegir realizar cursos sólo en ciertos periodo. Tales consideraciones convierten el problema en una nueva versión que provee de una mejor representación de la realidad (denominado GBACP).

En [2] se discuten diferentes algoritmos en la resolución de GBACP entre los que se destacan programación lineal entera, simulated annealing y búsqueda Tabu dinámica, aplicando los movimientos de trasladar un curso desde un periodo a otro, e intercambiar cursos, es decir, tomar dos cursos en diferentes periodos e intercambiarlos. A partir de los datos de los distintos algoritmos probados se concluyó que simulated annealing fue el que mostró los mejores y más rápidos resultados.

3.1. Instancias y benchmark

Tal como se mencionó anteriormente, el problema BACP ha sido incluido en CSPLib[3] en el cual se presentan tres intancias posibles de resolver, las cuales han sido ampliamente utilizadas para diversas pruebas de modelos y diferentes algoritmos.

Para el caso de GBACP en [12] se muestran distintas instancias, algunas de las cuales han sido tratadas en [2, 3].

3.2. Modelo Matemático

Tal como se ha señalado anteriormente, el problema ha sido cubierto por varios autores en la literatura, los cuales además de introducir diferentes nociones de balance en la asignación de los cursos, también han propuesto diferentes modelos. A continuación se describen las notaciones y variables de los distintos modelos propuestos.

3.2.1. Modelo de programación entera (IP)

Este modelo ha sido utilizado en diferentes publicaciones, entre las que se destacan [1, 2, 5, 15, 14], definiendo los siguientes parámetros de entrada:

m: Número de cursos

n: Número de periodos académicos

α_i: Número de créditos del curso i, $\forall\, i = 1, ..., m$

[3] http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob030/index.html

β: Mínimo de carga académica permitida por periodo

γ: Máximo de carga académica permitida por periodo

δ: Mínima cantidad de cursos permitida por periodo

ϵ: Máxima cantidad de cursos permitida por periodo

3.2.1.1. Variables de decisión

Se tienen las siguientes variables de decisión:

$$x_{ij} = \begin{cases} 1 & \text{si el curso } i\text{es asignado al periodo } j; \forall\ i = 1, ...m \,\forall\, j\ = 1, ..., n \\ 0 & \text{en caso contrario} \end{cases}$$

c_j: carga académica en el periodo $j\ \forall\ j = 1, ..., n$

c : máxima carga académica de todos los periodos

3.2.1.2. Función Objetivo

La función objetivo más común es la señalada en [1]:

1. $Min\ c = Max\{c_1, ..., c_n\}$: Minimizar la máxima carga académica de todos los periodos, al ser la carga total constante, cuando se reduce la mayor carga se lleva a que las cargas en los distintos periodos sean más homogéneas.

Otras funciones objetivo utilizadas en la literatura son las siguientes:

1. $Min\, z = Max\{c_1, ..., c_n\} - Min\{c_1, ..., c_n\}$ [14]: Minimizar la diferencia entre la carga académica máxima y la carga académica mínima de todos los periodos.
2. $Min\, z\ = Max\{|c_1 - \mu|, ..., |c_n - \mu|\}$ [10]: Añadiendo la restricción $\mu = \frac{\sum_{i=1}^{n} c_i}{n}$ que define la carga académica promedio, se busca minimizar la máxima diferencia entre las distintas cargas académicas y la carga académica promedio.
3. $Min\, z = \sum |c_j - \mu|$ [15]: Añadiendo la restricción $\mu = \frac{\sum_{i=1}^{n} c_i}{n}$ que define la carga académica promedio, se busca minimizar la suma de las diferencias absolutas de las cargas académicas respecto de la carga académica promedio.

3.2.1.3. Restricciones

Las restricciones que deben cumplirse son las siguientes:

1. $c_j = \sum_{i=1}^{m} \alpha_i \cdot x_{ij}\ \ \forall\ j = 1, ..., n$: carga por periodo académico j
2. $c_j \leq c \quad \forall\, j = 1, ..., n$: c es la mayor de todas las cargas académicas

3. $\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, ..., m$: todos los cursos i deben ser asignados a algún periodo j
4. $\sum_{i=1}^{m} x_{ij} \geq \delta \quad \forall\, j = 1, ..., n$: el número de cursos en un periodo j debe ser igual o mayor al mínimo de cursos requerido
5. $\sum_{i=1}^{m} x_{ij} \leq \epsilon \quad \forall j = 1, ..., n$: el número de cursos en un periodo j debe ser menor o igual al máximo de cursos requerido
6. $c_j \geq \beta \quad \forall\, j = 1, ..., n$: la carga académica del periodo j debe ser igual o mayor al mínimo requerido
7. $c_j \leq \gamma \quad \forall\, j = 1, ..., n$: la carga académica del periodo j debe ser menor o igual al máximo requerido
8. $x_{bj} \leq \sum_{r=1}^{j-1} x_{ar} \quad \forall\, j = 2, ..., n$: conjunto de restricciones que representan el caso en que un curso b tiene como prerrequisito el curso a

3.2.1.4. Modelos de satisfacción de restricciones

Además del modelo tradicional propuesto, en [5] se proponen dos modelos de satisfacción de restricciones, uno basado en el modelo que se acaba de presentar y propuesto en [1], y otro que modifica la representación de la variable de decisión x_{ij} propuesta anteriormente, quedando de la siguiente manera:

1. Variables de desición:

   $x_i = j$ si el curso i es asignado al periodo académico j $\forall\ i = 1, ...m \forall\, j\ = 1, ..., n$

   c_j: carga académica en el periodo $j\ \forall\ j = 1, ..., n$

   c : máxima carga académica de todos los periodos

2. Restricciones: en líneas generales son las mismas restricciones que el modelo anterior, a excepción de las siguientes:

   $c_j = \sum_{i \in cursos: x_i = j} \alpha_i$: la carga académica de un periodo será la suma de todos los créditos de los cursos asignados a ese periodo

   $x_a < x_b$: conjunto de restricciones que representan el caso en que un curso b tiene como prerrequisito el curso a. Se destaca el hecho de que esta restricción, difícil de implementar en el modelo anterior, resulta casi natural en este modelo

   $\forall\, j = 1, ..., n$; al menos (j, x_i, γ) y a lo más (j, x_i, ϵ): para todos los periodos académicos. A estos se debe asignar al menos el mínimo γ de cursos y a lo más el número ϵ de cursos

3.2.1.5. Otros modelos

Además de los modelos vistos anteriormente, se han propuesto varios modelos que combinan ideas de programación entera, programación con restricciones y la utilización de los programas OPL y CPLEX [4], añadiendo gran número de restricciones, siendo muchas de ellas auxiliares.

Por otro lado, en [2] se propone una nueva función objetivo para el problema generalizado GBACP basada en la minimización de las restricciones insatisfechas, dado que el criterio propuesto en [1] falla en distribuir los cursos de manera apropiada cuando se trata de problemas más generales. En GBACP se especifica que no siempre es posible cumplir con las restricciones impuestas por las preferencias de los profesores, haciéndose necesario relajar el problema y penalizar la función objetivo. A raíz de lo anterior, se considera la formulación del problema con programación lineal, que consta de una función bi-objetivo formulada a partir de una función $L(s)$, que está basada en las cargas académicas y las violaciones $R(s)$ de las preferencias de profesores, siendo s una posible candidata a solución. La función bi-objetivo queda como sigue:

$F_\ell(s) = w_1 L(s) + w_2 R(s)$**;** donde w_1 y w_2 son pesos asignados a $L_\ell(s)$ y $R(s)$ respectivamente.

En líneas generales se aprecia que $L(s)$, que ayuda a encontrar soluciones con distribuciones balanceadas, es definido como la distancia en $norma_\ell$ de las cargas académicas de una asignación s respecto de una distribución de cargas ideales α. Además, se incorpora una serie de restricciones concernientes a la $norma_\ell$ que dan como resultado un modelo mucho más robusto y complejo adaptado para situaciones más cercanas a la realidad[2].

3.3. Resumen

En este capítulo se realizó una revisión de los avances más importantes que se han logrado en la resolución de BACP. Además, se presentaron las principales instancias propuestas en la literatura que servirán para realizar futuras comparaciones de rendimiento. Por otro lado, se han mostrado los dos principales modelos que son utilizados actualmente para resolver el problema, los cuales son el modelo de programación lineal entera y el modelo de satisfacción de restricciones, añadiendo las distintas funciones objetivo que sirven como medida de calidad a la hora de establecer qué tan equilibrada es una solución. En general, los métodos que obtuvieron mejores resultados fueron aquellos que estaban basados en modelos híbridos de satisfacción de restricciones y programación lineal entera, que fueron resueltos beneficiándose de la propagación de restricciones. Si bien estos métodos han servido para solucionar el BACP en tiempos bajos, es preciso notar que fue necesario mezclar y complementar modelos para obtener

buenos tiempos, esto tiene la desventaja de que si el problema y sus modelos aumentan su complejidad no existe garantía de que la misma estrategia siga funcionando. Es más, en [2]se señala que el GBACP es intratable mediante los métodos de programación lineal entera y satisfacción de restricciones, señalando que aún es necesario desarrollar métodos heurísticos que puedan encontrar soluciones de buena calidad en tiempos razonables.

En el siguiente capítulo se presenta una nueva propuesta basada en Tabu Search, estableciendo las estrategias y algoritmos para la resolución de BACP, a la vez que se establece como un punto de partida para la resolución de GBACP.

4 Descripción del algoritmo

En esta sección, se presenta el enfoque utilizado para solucionar el problema BACP mediante la metaheurística de búsqueda Tabu Search. Para esto, se definen a continuación las distintas partes del algoritmo, así como también la generación de la solución inicial, las estrategias usadas para abordar el problema, los tipos de movimientos utilizados y las estrategias a seguir para la elección de las soluciones.

4.1. Tabu Search

Tabu Search es una técnica cuya principal idea es la utilización de memoria, implementada en una lista de movimientos prohibidos con el fin de evitar ciclos. De este modo, aquellos movimientos ya realizados recientemente son prohibidos durante un tiempo determinado por el largo de la lista; cuando la lista está llena y un nuevo elemento es añadido, el primer elemento ingresado es eliminado de la misma (olvidado) con lo que puede volver a visitarse. Por otro lado, Tabu Search elige siempre el mejor elemento del vecindario no Tabu, con lo que incorpora a la búsqueda la aceptación de elementos que pueden empeorar el valor de la función objetivo. El algoritmo termina cuando se ha alcanzado un máximo de iteraciones, cuando todos los movimientos del vecindario son Tabu o cuando se ha alcanzado algún otro criterio definido previamente. El Algoritmo 4.1 muestra el pseudocódigo genérico de Tabu Search.

Algoritmo 4.1 TabuSearch()

```
1  T=[]
2  S = SolucionInicial()
3  S_mejor = S
4  WHILE Criterio_Termino DO
5  BEGIN
6       V ← Vecindario(S)
7       s'← MejorSolucionPosible(V)
8       IF(f(s') > f(S_mejor))
9            S_mejor = s'
10 ␣␣␣␣S=s'
11      ActualizarListaTabu(T)
12 END
```

4.1.1. Representación

El siguiente modelo tiene inspiración en los modelos de satisfacción de restricciones, programación lineal entera e híbridos presentados en la literatura y se compone de la siguiente estructura ([1, 2, 5, 15, 14]):

Para representar una solución candidata, es decir, un curriculum compuesto de las asignaciones de todos los cursos a todos los períodos cumpliendo con las restricciones impuestas, se utiliza un arreglo donde sus índices representan un curso de la malla y su contenido es el período al cual está asignado dicho curso. Esto se muestra en la Figura 4.1.

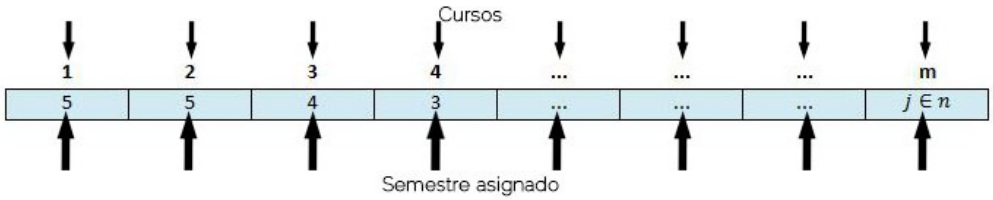


Figura 4.1: Representación de una solución como vector. Fuente: Elaboración propia.

Esta representación permite identificar fácilmente qué cursos están asignados a qué periodos además de satisfacer ciertas restricciones como que un curso debe realizarse sólo en un periodo específico. Es necesario recorrer todo el arreglo para chequear que todos los cursos hayan sido asignados a un periodo.

4.1.2. Generación de la solución inicial

La solución inicial que se genera se construye considerando las restricciones del problema, como consecuencia la solución inicial es siempre factible. Para lograr lo anterior se utiliza un algoritmo Greedy que consta de tres partes principales: la asignación de todos aquellos cursos que no posean prerrequisitos, la asignación progresiva de cursos cuyos prerrequisitos han sido cumplidos y finalmente una etapa de reparación que busca asignar cursos a períodos que aún no cumplen con las restricciones de cantidad mínima y máxima de cursos o carga académica. Una vez que una solución inicial factible ha sido alcanzada, se procede a aplicar un procedimiento para aleatorizarla con el fin de favorecer la exploración del algoritmo. El Algoritmo 4.2 muestra el pseudocódigo general para generar la solución inicial[1].

[1]Un video del proceso de generación de la solución inicial para una instancia de BACP con 8 periodos hasta antes de la aleatorización puede ser visto en http://youtu.be/-0YyDCcUSa0

Algoritmo 4.2 GenerarCurriculumSolucionInicial()

```
1 CurriculumSolucionInicial ← ∅
2 AsignarCursosSinprerrequisitos()
3 AsignarCursosprerrequisitosCumplidos()
4 AsignarCursosperíodosSinCursos()
5 AleatorizarSolucionInicial()
```

4.1.2.1. Asignación de cursos sin prerrequisitos

En esta sección del algoritmo se recorren todos los cursos existentes, en caso de que un curso no tenga prerrequisitos se asigna al primer período disponible, es decir, al primer período cuya asignación de dicho curso no viole las restricciones de mínimo y máximo de cursos por período, ni las de mínimo y máximo de créditos por período. Luego de la ejecución de este procedimiento se tendrá una malla curricular cuyos primeros períodos estarán con cursos que no tienen prerrequisitos. Lo anterior puede verse en el Algoritmo 4.3 [2].

Algoritmo 4.3 AsignarCursosSinPrerrequisitos()

```
1  FOR curso c IN cursos DO:
2  BEGIN
3      IF(!Tieneprerrequisitos(c) AND NoAsignado(c)) THEN
4      BEGIN
5          FOR período p IN períodos DO:
6          BEGIN
7              (*cumplir restricciones de créditos por período*)
8              IF(EsCreditosFactible(p+α_c)) THEN
9              BEGIN
10                 (*cumplir restricciones de cursos por período*)
11                 IF(EsCursosFactible(p+ 1)) THEN
12                 BEGIN
13                     AsignarCursoAperíodo(c,p)
14                     BREAK
15                 END
16             END
17         END
18     END
19 END
```

[2] Un video del proceso de asignación de cursos sin prerrequisitos para una instancia de BACP con 8 periodos puede ser visto en http://youtu.be/oIXsABCky2E

4.1.2.2. Asignación de cursos con prerrequisitos cumplidos

Luego del procedimiento anterior, del total de cursos existirán varios cuyos prerrequisitos fueron añadidos a la malla. A partir de esto, se procede a recorrer todos los cursos que si tienen prerrequisitos, y en caso de que todos sus prerrequisitos hayan sido cumplidos, se asignan al primer período cuya asignación no viole las restricciones de cursos y créditos mínimos y máximos y que sea posterior al período del último de los prerrequisitos de dicho curso. Este algoritmo es incremental, ya que a medida que se ejecuta se cumplirán los prerrequisitos necesarios para más cursos, lo que posibilitará que estos también puedan ser asignados. Luego de la ejecución de este procedimiento, la malla tendrá todos los cursos asignados y cumplirá con todas las restricciones de prerrequisitos, pero como los cursos fueron asignados a los primeros periodos que las restricciones le permitían, es posible que existan períodos sin cursos o que aún violen las restricciones de mínimo y máximo de créditos y cursos. El Algoritmo 4.4 muestra el pseudocódigo de esta sección [3].

[3]Un video del proceso de asignación de cursos con prerrequisitos cumplidos para una instancia de BACP con 8 periodos puede ser visto en http://youtu.be/64yRcIc4dYI

Algoritmo 4.4 AsignarCursosPrerrequisitosCumplidos()

```
1  (*flag:señala que hasta la iteración actual aún faltan cursos por asignar
       *)
2  faltanCursos ← true
3  WHILE (faltanCursos) DO
4  BEGIN
5       faltanCursos ← false
6       FOR curso c IN cursos DO:
7       BEGIN
8            IF(Tieneprerrequisitos(c) AND NoAsignado(c)) THEN
9            BEGIN
10                IF(prerrequisitosCumplidos(c)) THEN
11                BEGIN
12                     FOR período p IN períodos DO:
13                     BEGIN
14                          (*cumplir restricciones de créditos por período*)
15                          IF(EsCreditosFactible(p+α_c)) THEN
16                          BEGIN
17                               (*cumplir restricciones de cursos por período*)
18                               IF(EsCursosFactible(p+ 1)) THEN
19                               BEGIN
20                                    IF(p >períodoUltimoprerrequisito(c))
21                                         AsignarCursoAperíodo(c,p)
22                                         BREAK
23                               END
24                          END
25                     END
26                END
27                ELSE
28                     faltanCursos ← true
29           END
30      END
31 END
```

4.1.2.3. Asignación de cursos a periodos sin cursos

Tal como se señaló en la sección anterior, si bien todos los cursos han sido asignados y se cumplen sus prerrequisitos, pueden existir periodos que aún no cumplan con las restricciones de máximo y mínimo de créditos. Por lo anterior, este procedimiento tiene por finalidad reparar la solución en caso de que ésta aún sea infactible. Para esto se realiza un recorrido en todos los periodos, se toman aquellos que no cumplen las restricciones de créditos o cursos, y se les asignan cursos escogidos de tal manera que la asignación no viole las restricciones de prerrequisitos de la malla, ni tampoco las restricciones de cursos ni créditos mínimos y máximos del periodo al cual se le extrae el curso escogido. El criterio para escoger los cursos se ha diseñado para mejorar la

calidad de la solución inicial, es decir, minimizar la cantidad de créditos del período con más créditos. Es por esto que el ramo escogido será uno perteneciente al período con más créditos asignados. En caso de que ninguno de los cursos de este periodo puedan asignarse, se escoge el siguiente periodo con más carga. El Algoritmo 4.5 muestra el pseudocódigo de esta sección [4].

Algoritmo 4.5 AsignarCursosPeríodosSinCursos()

```
1  pMax ← CalcularperíodoMaxCarga()
2  FOR período p IN períodos DO:
3  BEGIN
4      (*período no cumple  con restricciones de cretidos o cursos*)
5      WHILE(!(EsCreditosFactible(p)) OR !(EsCursosFactible(p)) DO
6      BEGIN
7          FOR curso c IN cursosAsignados(pMax) DO
8              AsignarCursoAperíodo(c,p)
9              IF(EsprerrequisitosFactible(Curriculum) && EsCreditosFactible
                   (pMax) && EsCursosFactible (pMax)) DO
10             BEGIN
11                 (*Asignación correcta: recalcular período con más
                       creditos y asignar otro curso*)
12                 break
13             END
14             ELSE
15             BEGIN
16                 (*Asignación incorrecta: deshacer la asignación y tratar
                       con el siguiente ramo *)
17                 AsignarCursoAperíodo(c,pMax)
18             END
19         BEGIN
20         (*Buscar el período con más carga despues del ya escogido*)
21         pMax ← calcularSiguienteperíodoMaxCarga()
22     END
23 END
```

4.1.2.4. Aleatorización de la solución inicial

Una vez finalizado el algoritmo Greedy, se verifica si éste ha logrado obtener la solución óptima, en caso de haberla encontrado se termina el algoritmo. En caso contrario, se toma la solución inicial factible generada y se procede a generar otra haciendo una cantidad aleatoria de movimientos simples que mantengan la factibilidad. Vale destacar que cada uno de los movimientos aplicados no toman en cuenta el valor de la función objetivo, ya que su finalidad es lograr definir diversos puntos de partida diferentes

[4]Un video del proceso de asignación de cursos a periodos sin cursos para una instancia de BACP con 8 periodos puede ser visto en http://youtu.be/OColgvW78F8

antes de comenzar con la aplicación de Tabu Search. Para esto, se define inicialmente un número aleatorio de movimientos que se aplicarán sobre la solución inicial. Cada movimiento será a su vez hacer Swap entre dos cursos escogidos aleatoriamente de dos períodos académicos distintos, o insertar un curso de un período en otro. El Algoritmo 4.6 muestra el pseudocódigo de este procedimiento.

Algoritmo 4.6 AleatorizarSolucionInicial()

```
1  cantidadVecesAleatorizar ← generarNumeroAleatorio()
2  FOR i in RANGE (0,cantidadVecesAleatorizar) DO
3  BEGIN
4      (*Se escoge que tipo de movimiento se usará: 1 Swap, 2 Insert*)
5      tipoMovimientoAleatorio ← generarNumeroAleatorio(1,2)
6      IF (tipoMovimientoAleatorio=1)
7      BEGIN
8          WHILE (true) DO
9          BEGIN
10             (*Se escogen dos períodos distintos y de estos, dos cursos
                   para hacer el swap*)
11             período1 ← generarNumeroAleatorio(1,n)
12             período2 ← período1
13             WHILE (período2 = período1)
14                 período2 ← generarNumeroAleatorio(1,n)
15             curso1 ← generarNumeroAleatorio(1,cantidadRamosperiodo1)
16             curso2 ← generarNumeroAleatorio(1,cantidadRamosperiodo2)
17             SwapCursos(CurriculumSolucionInicial,curso1,curso2)
18             IF(EsFactible(CurriculumSolucionInicial))
19                 BREAK
20             ELSE
21                 (*Se realiza un swap para volver al estado anterior*)
22                 SwapCursos(CurriculumSolucionInicial,curso1,curso2)
23         END
24     END
25     ELSE
26     BEGIN
27         WHILE (true) DO
28         BEGIN
29             (*Se escogen dos períodos distintos y de uno de estos, un
                   curso para insertarlo en el otro período*)
30             período1 ← generarNumeroAleatorio(1,n)
31             período2 ← período1
32             WHILE (período2 = período1)
33                 período2 ← generarNumeroAleatorio(1,n)
34             curso1 ← generarNumeroAleatorio(1,cantidadRamosperiodo1)
35             InsertCurso(CurriculumSolucionInicial,período2,curso1)
36             IF(EsFactible(CurriculumSolucionInicial))
37                 BREAK
38             ELSE
39                 (*Se realiza Insert para volver al estado anterior*)
40                 InsertCurso(CurriculumSolucionInicial,período1,curso1)
41         END
42     END
43 END
```

4.2. Tabu Search para BACP

Una vez que se tiene una solución inicial factible se procede a aplicar Tabu Search, para esto se ha adaptado la estructura genérica del algoritmo mostrado anteriormente a una que es capaz de solucionar BACP. En el algoritmo propuesto se trabajará con factibilidad, es decir, las soluciones candidatas serán siempre factibles al igual que en la lista Tabu. El criterio de término definido es el de máximo número de iteraciones, cuando no existan posibles movimientos a realizar debido a la lista Tabu o cuando se ha alcanzado el óptimo global de cantidad de créditos en los periodos de la carrera (de acuerdo a la ecuación presentada en la Subsubsección 4.2.3.1). Antes de presentar el algoritmo se describe cada uno de sus componentes.

4.2.1. Movimiento

El movimiento escogido tiene como base el hecho de que se aplica sobre una solución factible generada con anterioridad, y es una mezcla de dos movimientos más simples, cada uno genera un semivecindario y en conjunto generan el vecindario de soluciones de una solución específica. Estos movimientos son listados a continuación:

1. **Insert de cursos:** Se toma un curso de un período y se inserta en otro. Este movimiento puede generar soluciones no factibles, pero tiene la ventaja de brindar variabilidad y con ello otorgar exploración al algoritmo, ya que permite aumentar o disminuir la cantidad de cursos (y por ende también los créditos) de un periodo. El tamaño de cada semivecindario generado por este movimiento es de $T = (CantidadSemestres - 1) * CantidadCursos = (n-1) * m$. La Figura 4.2 muestra la generación de un semivecindario utilizando este movimiento.

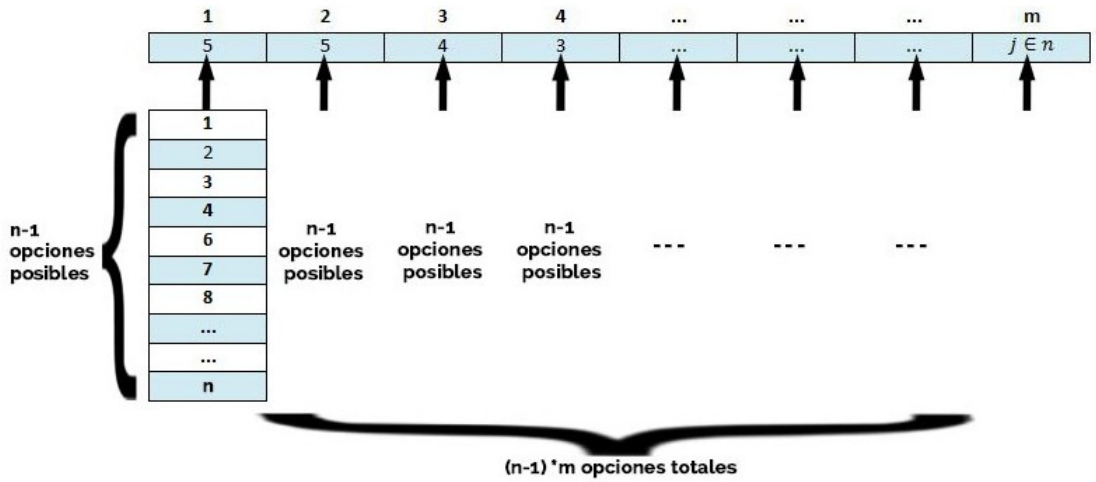


Figura 4.2: Generación de un semivecindario a partir del movimiento Insert. Fuente: Elaboración propia.

2. **Swap de cursos:** Tomar dos cursos asociados a distintos periodos y asignar cada uno al periodo contrario. Este movimiento puede generar soluciones no factibles, pero asumiendo que se parte de una solución actual factible, se generarán siempre soluciones que cumplen con las restricciones de mínimo y máximo de cursos en cada periodo. Por otro lado, este movimiento es capaz de cambiar el valor de la cantidad de créditos asignados a dos periodos al mismo tiempo, lo que incorpora una mayor explotación al ser aplicado. El tamaño de los vecindarios generados por este movimiento, debido a su naturaleza combinatoria, es más pequeño que los generados por Insert, variando su tamaño a través del tiempo dependiendo de la cantidad de cursos que estén asignados a cada periodo. La Figura 4.3 muestra la generación de un semivecindario utilizando este movimiento.

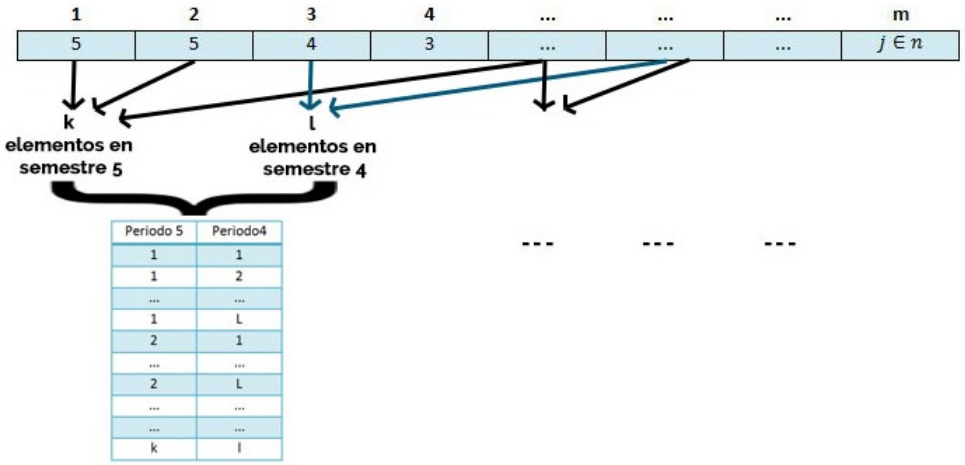


Figura 4.3: Generación de un semivecindario a partir del movimiento Swap. Fuente: Elaboración propia.

A continuación, el vecindario del movimiento general se genera a partir de la generación de los dos semivecindarios.

4.2.2. Lista Tabu

Debido a la naturaleza dual del movimiento, la lista Tabu se compone de dos sublistas, una para los movimientos Insert y otra para movimientos Swap. Cada vez que el algoritmo elige uno de los dos movimientos como siguiente a aplicar, lo agrega a la lista correspondiente. La comprobación de movimientos prohibidos se hace de forma independiente para cada tipo movimiento, teniéndose que un conjunto de movimientos, cualquiera sea su órden de aplicación, estarán almacenados de la misma forma en que se aplicaron en las listas separadas, evitando de esta manera volver hacia puntos ya visitados. La Figura 4.4 y la Figura 4.5 muestran la estructura de arreglo de cada una de las listas.

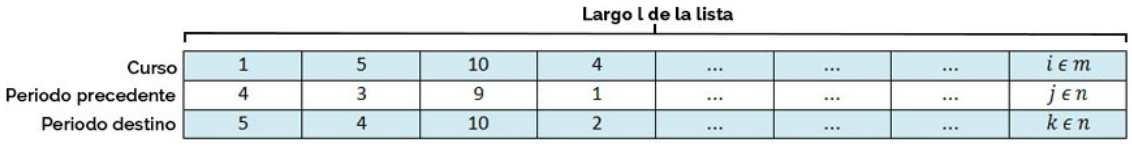


Figura 4.4: Lista Tabu de largo l de movimientos Insert. Fuente: Elaboración propia.

Largo l de la lista

Curso precedente	1	5	10	4	...	...	...	$i \epsilon m$
Curso destino	2	1	4	8	...	...	...	$l \epsilon m$
Periodo precedente	4	3	9	1	...	...	...	$j \epsilon n$
Periodo destino	5	4	10	2	...	...	...	$k \epsilon n$

Figura 4.5: Lista Tabu de largo l de movimientos Swap. Fuente: Elaboración propia.

4.2.3. Función objetivo, función de evaluación y criterio de selección

4.2.3.1. Función objetivo del problema

La función objetivo escogida es la máxima carga académica de todos los periodos introducida en [3], por lo que el algoritmo se basa en la minimización de esta función. Tal como se ha señalado en [1], es posible verificar si se ha alcanzado un óptimo calculando el óptimo teórico asociado al problema mediante la siguiente ecuación:

$$Óptimo_{ideal} = \left\lceil \frac{CantidadCreditosTotales}{CantidadSemestres} \right\rceil$$

4.2.3.2. Función de evaluación y criterio de selección de soluciones

La función de evaluación propuesta es la suma de la función objetivo más la sumatoria de las diferencias absolutas con respecto a la media de la carga académica de todos los períodos, mostrada en la sección 3.2.1.2. A partir de esto, la función de evaluación se muestra en la siguiente ecuación:

$$Fit = Max\{c_1, ..., c_n\} + \sum_{j=1}^{n} |c_j - \mu|$$

Donde c_j es la carga académica en el período $j \ \forall \ j = 1, ..., n$

y μ es la carga académica promedio calculada apartir de $\mu = \frac{\sum_{i=1}^{n} c_i}{n}$

La razón principal de la elección de esta función es debido al criterio de selección de soluciones. Tabu Search es una metaheurística que cambia siempre al vecino que tiene el mejor valor de evaluación. Si bien la función $Min\{max\}$ (minimizar el periodo con máxima carga) sirve en la búsqueda del óptimo y ha sido utilizada como un criterio

para escoger soluciones en [1, 5, 6], tiene problemas a la hora de escoger una solución porque es muy probable que en el vecindario existan muchas soluciones con el mismo valor de función. Como es preciso escoger una sola solución, de no incorporarse otra heurística para escoger se elegirá alguno de los candidatos al azar. Este problema se incrementa en la medida en que se presenta una mayor cantidad de vecinos con el mismo valor. A partir de esto, la función de diferencia absoluta incorpora dos ventajas: por un lado, se tiene un elemento de dispersión estadística que al buscar soluciones con menor valor de evaluación genera soluciones más uniformes, y por otro, al ser un valor decimal se tiene un indicador más fino a la hora de escoger soluciones cuando $Min\{max\}$ produce muchas soluciones con el mismo valor. En definitiva, el objetivo será discriminar de buena manera entre las distintas soluciones candidatas y guiar de mejor manera al algoritmo que si lo hiciera utilizando sólo $Min\{max\}$.

Por último, cuando existen soluciones que poseen el mismo valor de evaluación dentro de un semivecindario, se genera un número al azar que permite escoger una solución dentro del mismo. Posteriormente la solución escogida por Tabu Search como mejor vecino es aquella que tiene el valor de evaluación más pequeño de entre las dos escogidas en los dos semivecindarios. De existir dos soluciones con el mismo valor, se escoge otro número al azar que indica cual de las dos escoger. El Algoritmo 4.7 muestra el pseudocódigo de la generación de vecindarios y el Algoritmo 4.8 muestra la elección de la mejor solución señalada anteriormente.

Algoritmo 4.7 GenerarVecindario()

```
1 (*Vecindario compuesto de todas las posibles inserciones de un curso de
      un período en los demás*)
2 VecindarioInsert ← GenerarVecindarioInsert()
3 (*Vecindario compuesto de todos los posibles swap entre dos cursos de
      períodos distintos*)
4 VecindarioSwap ← GenerarVecindarioSwap()
5 Vecindario ← VecindarioInsert + VecindarioSwap
```

Algoritmo 4.8 EscogerMejorMovimientoNoTabuVecindario()

```
1 (*En caso de muchas soluciones con igual valor, se escoje una al azar*)
2 BuscarMejorMovimientoNoTabuInsert()
3 (*En caso de muchas soluciones con igual valor, se escoje una al azar*)
4 BuscarMejorMovimientoNoTabuSwap()
5 (* En caso de que ambas soluciones tengan igual valor, se escoje una al
      azar)
6 BuscarMejorMovimientoNoTabuSwapInsert()
```

4.2.4. Algoritmo propuesto

Una vez teniendo definidas todas las partes del algoritmo, en el Algoritmo 4.9 se muestra el pseudocódigo de Tabu Search para BACP.

Algoritmo 4.9 TabuSearchBACP()

```
1  (*Solución inicial factible*)
2  CurriculumActual ← GenerarCurriculumSolucionInicial()
3  maxIteraciones ← máximo número de iteraciones.
4  CurriculumMejorSolucion ← CurriculumActual
5  Iteraciones ← 0
6  ListaTabu ←∅
7  WHILE (iteraciones < maxIteraciones) DO
8  BEGIN
9       Vecindario ← GenerarVecindario()
10      IF(all Vecindario in ListaTabu) (*Todos los movimientos del
            vecindario son Tabu*)
11          BREAK
12      else
13      BEGIN
14          MejorMovimiento ← EscogerMejorMovimientoNoTabuVecindario()
15          MejorSolucion ← AplicarMovimiento(MejorMovimiento)
16          ListaTabu += MejorMovimiento
17          IF(CurriculumActual < MejorSolucion)
18              MejorSolucion ← CurriculumActual
19          IF(CurriculumActual = óptimo)
20              BREAK
21          Iteraciones +=1
22      END
23 END
24 RETURN CurriculumMejorSolucion
```

4.3. Conclusiones

En este capítulo se ha presentado una metaheurística basada en Tabu Search para resolver el problema de diseño de mallas curriculares estableciendo la representación de las soluciones, movimientos que se utilizarán y cómo estos fueron elegidos para permitir la explotación y exploración del algoritmo. Además se ha presentado la función objetivo utilizada y qué criterio será empleado para definir si se ha encontrado un óptimo. Por otro lado, también se ha mostrado cuál será la función de evaluación que servirá para medir la calidad de las soluciones elegidas y con ello guiar al algoritmo en la consecución de mejores soluciones. Uno de los aspectos que se tomó en cuenta es contar con una diversidad de soluciones iniciales, para ello se propuso un algoritmo

Greedy cuyos componentes están orientados a generar soluciones candidatas factibles y que son posteriormente perturbadas para producir la variabilidad deseada.

Se puso especial atención en la definición de una función de evaluación que permitiera guiar al algoritmo en la búsqueda de soluciones que optimicen la carga académica, para ello se incorporó una medida de dispersión cuyo objetivo es medir la homogeneidad de las soluciones candidatas y permitir una mejor discriminación de las soluciones. El algoritmo trabaja sólo con soluciones factibles por lo que se definieron dos movimientos especialmente diseñados para el problema. En el siguiente capítulo, se mostrarán los resultados experimentales al aplicar la metaheurística diseñada en el capítulo 4 para las instancias de BACP8, BACP10 y BACP12 y para un conjunto de cien instancias generadas a partir de éstas, además de presentar las comparaciones de dichos resultados con los expuestos hasta ahora en la literatura.

5 Resultados Experimentales

En este capítulo se presentan las diferentes instancias y los experimentos realizados, los cuales permitirán observar los resultados de la aplicación de la metaheurística basada en Tabu Search presentada en el capítulo 4 aplicada a estas instancias específicas. Además de esto se hace una comparación con los distintos resultados publicados en la literatura y una discusión de características importantes de la metaheurística así como también su efectividad, eficiencia, y la influencia de la elección de los parámetros en los resultados obtenidos.

5.1. Especificaciones del equipo de trabajo

Las pruebas fueron realizadas en un equipo con las siguientes características:

1. Sistema operativo: Ubuntu 12.04.10
2. Procesador: Intel core i3 2.53 GHz. Debido a que el algoritmo no fue hecho para trabajar con múltiples hilos, sólo se utiliza un core.
3. Memoria RAM 6.0 GB (5.74 utilizables).

5.2. Métodos de evaluación

Tal como se mencionó anteriormente el algoritmo trabaja minimizando la función objetivo, la cual corresponde a la carga máxima del periodo con más carga. Para evaluar la calidad de las soluciones el algoritmo incorpora una medida de dispersión obteniéndose que la función de fitness es la suma de la función objetivo y la sumatoria de las diferencias absolutas con respecto a la media de la carga académica de todos los períodos.

5.3. Experimentos

Para la realización de los experimentos se utilizaron tres instancias de BACP publicadas en CSP lib [1], las cuales constan de 8, 10 y 12 periodos respectivamente, estas

[1] http://www.cs.st-andrews.ac.uk/~ianm/CSPLib/prob/prob030/index.html

instancias han sido ampliamente utilizadas en la literatura y fueron propuestas inicialmente por [1]. Además de lo anterior se utilizarán las instancias de prueba creadas en [11] provistas en `http://becool.info.ucl.ac.be/resources/bacp`, las cuales son 100 instancias obtenidas mediante la edición de la instancia de 12 periodos del CSP lib, cambiando aleatoriamente la carga de algunos cursos dejándolos con un peso entre 1 y 5, y conservando 50 de los 65 prerrequisitos totales.

Las características generales de las instancias de prueba utilizadas se muestran en la Figura 5.1.

Instancia	Min # cursos	Max # cursos	Min carga	Max carga	# Periodos	# Cursos	# Prerreq.
Bacp8	2	10	10	24	8	46	38
Bacp10	2	10	10	24	10	42	34
Bacp12	2	10	10	24	12	66	65
inst0-100	5	7	10	24	12	66	50

Figura 5.1: Características de las instancias de prueba. Fuente: Elaboración propia.

Los parámetros que se variaron fueron: el largo de la lista Tabu, la cantidad máxima de iteraciones antes de terminar el algoritmo y las semillas para generar números aleatorios. Dado que las distintas soluciones iniciales se generan aleatorizando una solución creada previamente mediante un algoritmo Greedy, cada semilla generó una solución inicial distinta. La Figura 5.2 resume los parámetros usados para cada instancia y la cantidad de veces totales que se ejecutó el programa con cada una.

Instancia	Largo de lista tabú	Cant. It. máxima	Cant. Semillas	Cant. Ejecuciones
Bacp8	10, 50, 100,200,500	30,50,100,300,500,800	21	330
Bacp10	10, 50, 100,200,500	30,50,100,300,500,800	21	330
Bacp12	10, 50, 100,200,500	30,50,100,300,500,800	21	330
inst0-100	100,200,500	100,300,500	8	36 (cada instancia)

Figura 5.2: Parámetros del algoritmo utilizados en los experimentos. Fuente: Elaboración propia.

Por otro lado en los resultados se registraron la calidad de las soluciones iniciales, la cantidad de iteraciones que el algoritmo realizó antes de terminar, si obtuvo o no el óptimo y tiempo empleado en la búsqueda. Además de lo anterior se analiza la capacidad que el algoritmo tiene para encontrar o no la solución óptima y la dificultad que tuvo en alcanzarlo.

5.4. Resultados

Las pruebas se ejecutaron en dos lotes, primero se ejecutaron las instancias Bacp8, Bacp10 y Bacp12 y luego las 100 instancias.

5.4.1. Instancias CSP Lib

Las instancias del CSP Lib han sido ampliamente utilizadas para experimentación en la literatura. Desde su propuesta en [1], diferentes estrategias y búsquedas han sido usadas logrando algunas el valor óptimo de cada una. A modo de referencia se muestran los distintos resultados de la literatura.

Fuente	Calidad mejor sol	T [s]
Bacp8		
[1]	17 (óptimo)	1459,73
[5]	17 (óptimo)	0,29
[4]	17 (óptimo)	0,38
[6]	17 (óptimo)	15,05
[9]	17 (óptimo)	1,248
Bacp12		
[1]	sin resultados	-
[5]	17 (óptimo)	1,09
[4]	17 (óptimo)	1,20
[6]	18 (no óptimo)	35,20
[9]	18 (no óptimo)	6,368

Fuente	Calidad mejor sol	T [s]
Bacp10		
[1]	24 (no óptimo)	1626,84
[5]	14 (óptimo)	0,59
[4]	14 (óptimo)	0,36
[6]	14 (óptimo)	34,84
[9]	14 (óptimo)	1,949

Entre los objetivos principales de esta memoria, está el lograr resolver estas instancias y a partir de esto, comparar los resultados con los mejores expuestos en la tabla anterior, los cuales han sido obtenidos por [5] y [4] mediante el uso de programación con restricciones y programación lineal entera, beneficiándose especialmente del uso de la propagación de restricciones. A continuación de presentan los resultados obtenidos para las mismas instancias, como se señaló anteriormente cada una se ejecutó 330 veces, con el fin de lograr resultados confiables sobre las características de la metaheurística. En la Figura 5.3 se muestra el porcentaje de ejecuciones que lograron alcanzar el valor de carga óptima teórica.

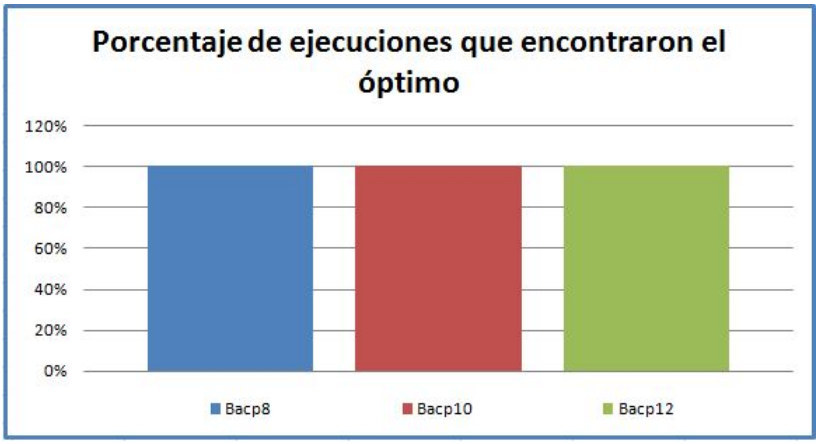


Figura 5.3: Gráfico del porcentaje de las 330 ejecuciones para las instancias Bacp8, Bacp10 y Bacp12 que encontraron el óptimo. Fuente: Elaboración propia.

Si bien se realizaron una gran cantidad de ejecuciones por cada instancia, se tiene que el algoritmo pudo encontrar el óptimo en cada una de ellas para todas las ejecuciones. Esto señala que no importando los parámetros asignados el algoritmo, éste tiene una ***alta eficacia*** para estas instancias. Es preciso destacar que además de su capacidad para encontrar soluciones óptimas el algoritmo trabaja siempre con soluciones factibles, esto señala que la calidad de las soluciones es desde un principio aceptable.

A continuación en la Figura 5.4 se presenta un gráfico que muestra una comparación entre la calidad de las soluciones iniciales y la calidad del valor óptimo.

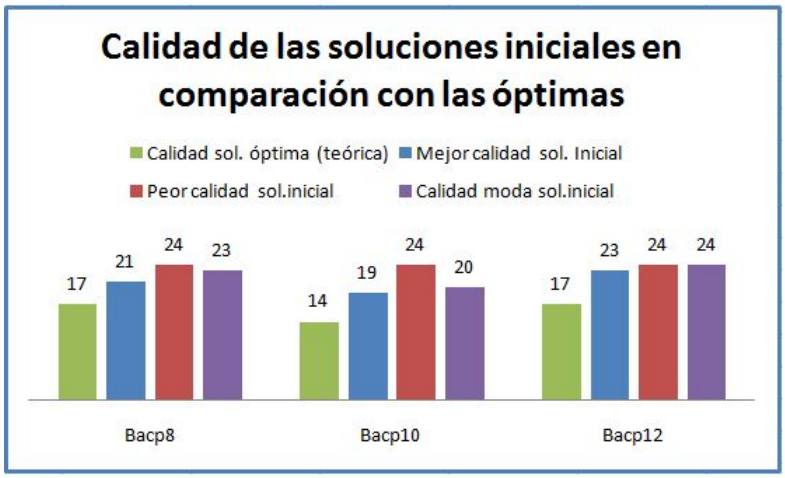


Figura 5.4: Gráfico de comparación de la calidad de la solución óptima teórica y calidad de soluciones iniciales mejor, peor y moda. Fuente: Elaboración propia.

Se aprecia que en las tres instancias el algoritmo Greedy pudo generar soluciones

iniciales relativamente aceptables en términos de calidad, se estima que la razón de esto es porque este algoritmo busca generar una solución inicial *factible* y seguidamente, se cumple con las restricciones de máximo y mínimo de carga académica por periodo, causa de esto la peor calidad de las soluciones iniciales siempre será 24 (carga máxima de cada periodo). Por otro lado la moda señala que tanto para Bacp8 como Bacp10 la mayor frecuencia en la calidad de las soluciones iniciales está por debajo de la cota superior, característica que no se repite para Bacp12 donde la mayor parte de las soluciones iniciales fueron de valor 24. Se estima que lo anterior se debe a la mayor dificultad de Bacp12 asociada a poseer una mayor cantidad de cursos (y con ello de carga) que debe distribuirse entre los distintos periodos. Además de lo anterior las barras que muestran las mejores calidades de las soluciones iniciales indican que de vez en cuando el algoritmo generó valores bastante buenos, cercanos a la mitad entre la cota superior y el valor óptimo. Al igual que antes esta característica no se cumple para Bacp12 debido a su dificultad intrínseca.

A continuación la Figura 5.5 muestra la cantidad de iteraciones en el peor y mejor caso para encontrar el óptimo su correspondiente moda para todas las ejecuciones.

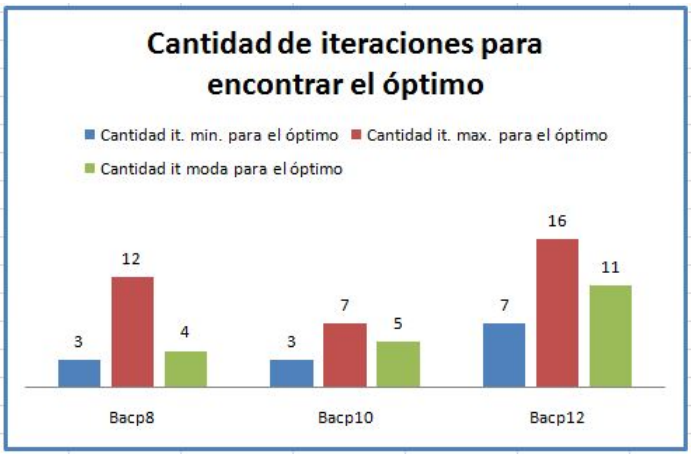


Figura 5.5: Gráfico de cantidad de iteraciones necesarias para encontrar el óptimo en los casos peor y mejor y moda de iteraciones necesarias. Fuente: Elaboración propia.

En el gráfico de la figura, se aprecia que a nivel global en las tres instancias la cantidad de iteraciones fue muy baja llegando incluso a las 7 iteraciones en el peor de los mejores casos, y 16 iteraciones en el peor de los peores casos. Estas características señalan que el algoritmo tiene una alta eficacia, es decir, que la cantidad de iteraciones necesarias para encontrar al óptimo son generalmente bajas. Al igual que el gráfico de la Figura 5.4 sobre calidad de soluciones iniciales, la cantidad de iteraciones señala que Bacp12 es la instancia más difícil, teniendo valores más altos en todos los casos. Por otro lado, se aprecia que Bacp10 es la instancia más fácil de resolver ya que tiene los valores

más bajos en todos los casos, la razón de esto es porque esta instancia posee incluso menos cursos que Bacp8, además que tener que distribuirlos en una mayor cantidad de periodos, esta característica tiene como consecuencia que exista un área de soluciones óptimas mayor y con ello que ésta sea más fácil de alcanzar.

En el gráfico de la Figura 5.6, se muestra un resumen de los tiempos requeridos para alcanzar el óptimo en todas las ejecuciones

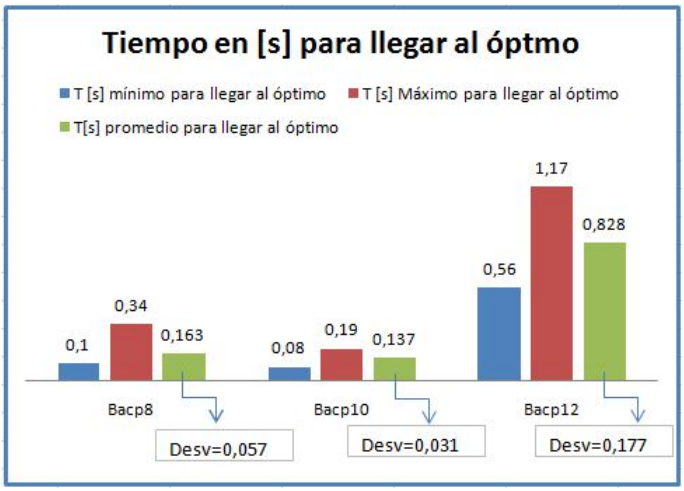


Figura 5.6: Gráfico del tiempo necesario para alcanzar el óptimo en el caso peor, mejor y promedio. Fuente: Elaboración propia.

En el gráfico mostrado puede verse que los patrones de dificultad se corroboran, es decir, Bacp10 es la instancia más sencilla de resolver ya que tiene menores tiempos de cómputo para todos los casos. Asimismo, Bacp12 es la instancia que toma más tiempo de cómputo para obtener el óptimo. Es preciso destacar que en esta instancia el tiempo utilizado en todas las situaciones es al menos tres veces mayor que en las otras dos instancias, se estima que esto se debe a que el tamaño de los vecindarios a generar para los movimientos swap e insert del algoritmo, son notablemente más grandes al poseer una mayor cantidad de cursos y periodos, de hecho tal como se señaló en el capítulo 4 de descripción del algoritmo, sólo el movimiento insert genera un vecindario de tamaño $T = (CantidadSemestres - 1) * CantidadCursos$. El valor pequeño de las desviaciones estándar señala que en general los tiempos no estuvieron alejados del promedio, por lo que éste puede verse como un buen indicador de tendencia. Además de lo anterior, se destaca que estos resultados son comparables con los mejores resultados de la literatura los cuales son [5] y [4]. Para [4] específicamente, se tiene que los mejores tiempos obtenidos fueron 0,38 [s]; 0,36 [s] y 1,20 [s] para Bacp8, Bacp10 y Bacp12 respectivamente. Tomando en cuenta que su equipo de trabajo constaba de un intel pentium III de 1.17 GHz de procesador con 512 MB de RAM estos tiempos

equiparables a 0.19 [s]; 0,18 [s] y 0,6 [s] respectivamente (ya que el procesador de trabajo para los experimentos actuales es aproximadamente el doble de rápido). Tomando en cuenta la gran diferencia entre la memoria RAM (6 GB vs 512 MB) y las generaciones distintas de los procesadores, es posible afirmar que los tiempos son aproximadamente iguales, ***esto sugiere que los resultados obtenidos en la presente memoria son equiparables a los mejores resultados de la literatura.***

5.4.2. 100 instancias obtenidas de Bacp12

Como se mostró anteriormente, este conjunto de instancias ha sido obtenido a partir de Bacp12, y entre sus características principales destaca que la cantidad de prerrequisitos fue reducida desde 65 a 50 (quitando aleatoriamente los que se eliminaron). Si bien el autor no señala la razón de este cambio, se estima que se hizo pensando en la fase de transición, es decir, reducir la cantidad de restricciones a una cantidad "media" para encontrar aquellas derivaciones de Bacp12 que tuviesen mayor dificultad.

Dado que la cantidad de instancias totales es muy grande como para mostrar los resultados de cada una, se decidió agruparlas de acuerdo a su valor óptimo de carga académica teórico. A partir de esto se presenta la Figura 5.7 que compara la cantidad de veces que se encontró el óptimo con la cantidad de ejecuciones totales de cada grupo.

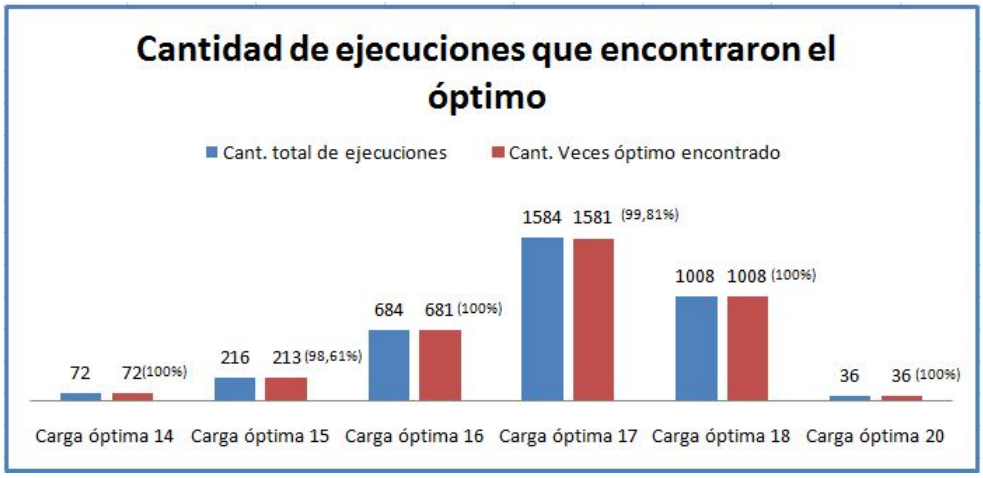


Figura 5.7: Gráfico de la cantidad y porcentaje de las ejecuciones totales que encontraron el óptimo, agrupadas por óptimo teórico. Fuente: Elaboración propia.

Al igual que en el gráfico de la Figura 5.3 se observa que el algoritmo es muy eficaz teniéndose que el mínimo valor de ésta en todos los grupos de instancias es de 98,61 %. Es más, de las 3600 ejecuciones totales de todas las instancias, el algoritmo no encontró el óptimo sólo en 9 oportunidades. En conjunto con lo anterior se aprecia también que

si bien se ha intervenido la instancia original otorgando a cada curso una carga de entre 1 y 5 *aleatoriamente,* la mayor cantidad de instancias tienen un valor óptimo teórico igual que la instancia original (carga académica de 17). Esto sugiere que podría evaluarse una técnica distinta para generar soluciones diferentes aunque es igualmente destacable la obtención de 684 y 1008 instancias cuyos valores óptimos teóricos son 16 y 18 correspondientemente.

A continuación en la Figura 5.8 se presenta un gráfico que muestra una comparación de la calidad de las soluciones iniciales.

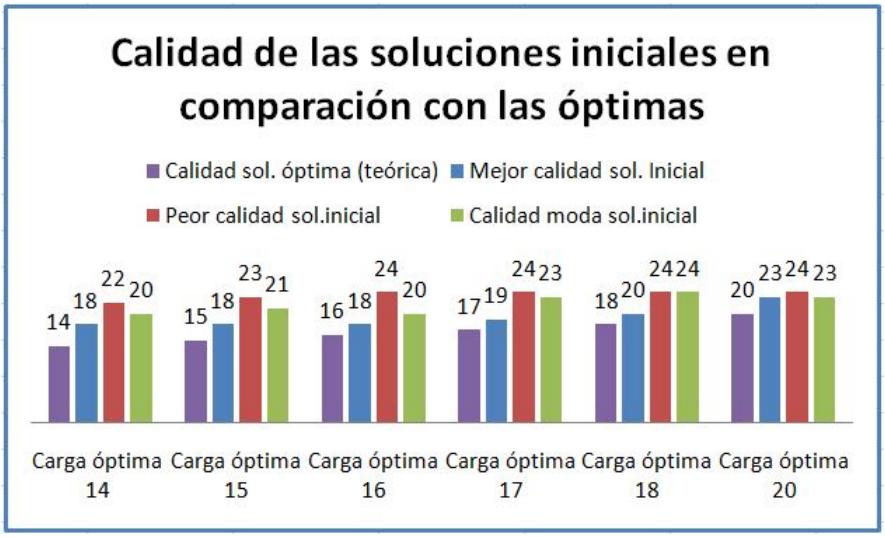


Figura 5.8: Gráfico de comparación de la calidad de la solución óptima teórica y calidad de soluciones iniciales en los casos mejor, peor y moda. Fuente: Elaboración propia.

En concordancia con los resultados de las instancias anteriores se aprecia que el algoritmo que genera las soluciones iniciales obtuvo calidades aceptables, teniéndose que en la mayoría, la mejor calidad de solución fue incluso mejor a la mitad de la diferencia entre el límite superior y el óptimo teórico. De la misma manera, la moda muestra que en la mayor parte de las ocasiones la solución inicial generada tiene mejor valor que la cota superior. Por último, la mejor calidad, peor calidad y moda muestran que las instancias más difíciles, al menos en términos de generar una solución inicial de buena calidad mediante el Greedy, son aquellas cuyo valor óptimo de carga académica es 17, 18 y 20.

La Figura 5.9 muestra la cantidad de iteraciones que ejecutó el algoritmo antes de encontrar el óptimo.

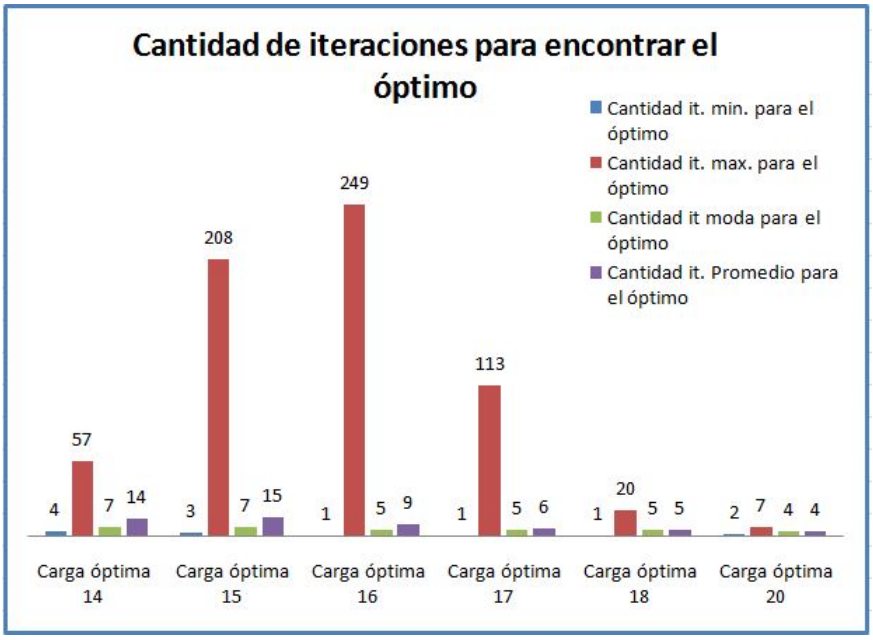


Figura 5.9: Gráfico de cantidad de iteraciones necesarias para encontrar el óptimo para los casos mejor, peor, moda y promedio. Fuente: Elaboración propia.

Observando la cantidad de iteraciones que el programa requirió para encontrar el óptimo puede afirmarse que de manera global el algoritmo resolvió muy rápido la totalidad de las instancias. Lo anterior está principalmente respaldado al observar las cargas mínimas, moda y promedio de iteraciones. A partir de esto se tiene que, salvo contadas ocasiones específicas, el algoritmo requirió 15 iteraciones o menos para resolver cualquier instancia. Observando en detalle los datos experimentales se corroboró esta idea, notando que el valor 57 del grupo de carga óptima 14 se logro 9 veces en una instancia específica y a partir de una solución inicial específica, de la misma se confirmó que los valores 208, 249 113 y 20 se obtuvieron por la misma situación.

En el gráfico de la Figura 5.10, se muestra un resumen de los tiempos requeridos para alcanzar el óptimo en todas las ejecuciones.

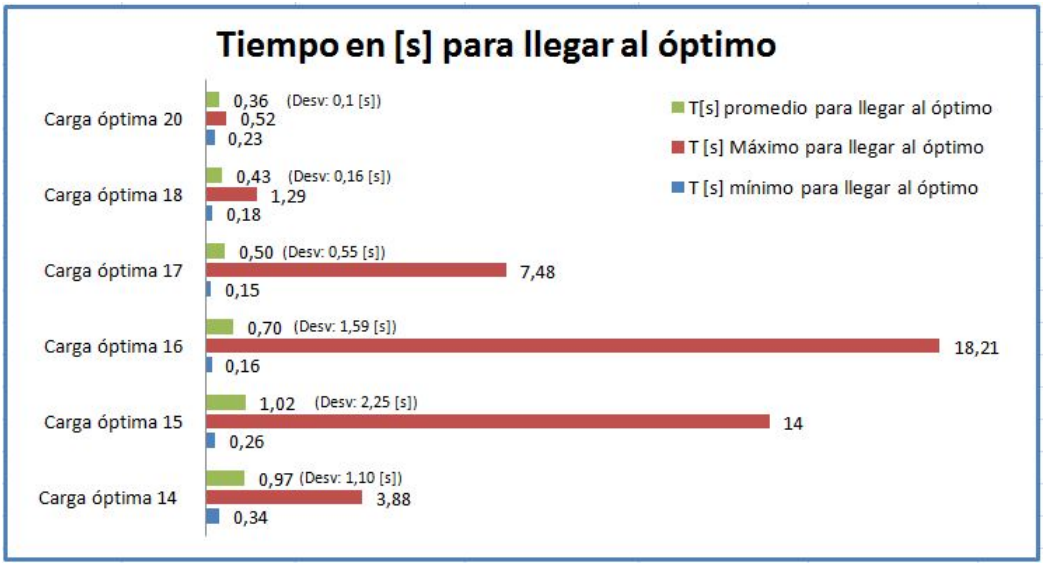


Figura 5.10: Tiempo necesario para alcanzar el óptimo. Fuente: Elaboración propia.

Este gráfico confirma todos los resultados obtenidos anteriormente, es decir, que el algoritmo solucionó las instancias rápidamente. Observando los valores de las desviaciones estándar se aprecia que para los grupos de carga óptima 14, 15 y 16 ésta es relativamente grande lo cual señala que los datos están dispersos alrededor del promedio, y se debe a que en ciertas ocasiones el algoritmo demoró muchísimo en comparación con lo normal y esto desvió el promedio hacia arriba. Esta característica es la misma que la vista en el gráfico de la Figura 5.9 (muchas iteraciones en los peores casos pero pocas iteraciones en la moda y caso promedio) y en conjunto, a partir de los datos anteriores y los de este gráfico puede decirse que en general el algoritmo, salvo contadas excepciones, demoró muy poco en encontrar la solución óptima. ***Esto confirma la idea de que la metaheurística es eficiente para resolver problemas BACP***.

5.5. Conclusiones

En este capítulo se mostraron los resultados de la aplicación de la metaheurística basada en Tabu Search a dos conjuntos de instancias, obteniendo muy buenos resultados. En el caso de las instancias de CSP Lib pudieron resolverse todas en pocas iteraciones y con poco tiempo de cómputo obteniéndose incluso resultados equiparables a los mejores resultados mostrados en la literatura. Por otro lado de los resultados de las 100 instancias basadas en Bacp12 pudo corroborarse que el algoritmo resuelve instancias de BACP en poco tiempo, con alta eficacia y eficientemente, propiedades deseables en una metaheurística [7].

Se estima que la principal fuente de éxito del algoritmo estuvo basada primero, en la facultad que tiene Tabu Search para recordar los movimientos y así permitir evitar ciclos y segundo en las estrategias de generación de una solución inicial factible mediante un algoritmo Greedy y la posterior aplicación de los movimientos en base a dos sub movimientos, los cuales en conjunto brindaron un vecindario diverso del cual escoger las mejores soluciones. Además, se concluye que la medida de dispersión agregada para generar la función de evaluación en el algoritmo tuvo un gran impacto a la hora de encontrar soluciones óptimas ya que en pruebas anteriores a la incorporación de ésta el algoritmo quedaba a menudo estancado en valores cercano al óptimo ya que existía una gran cantidad de soluciones que tenian ese mismo valor.

En el siguiente capítulo se mostrará una herramienta construida para mejorar el diseño de metaheurísticas que busquen solucionar BACP, destacando las ventajas que permite el uso de ésta y cómo la visualización de los datos puede permitir un rápido análisis de los mismos en favor de probar diferentes estrategias a seguir para resolver el problema.

6 Herramienta de visualización para metaheurísticas que solucionen BACP

En este capítulo se presenta una herramienta de visualización de datos, la cual permite ayudar en el diseño de metaheurísticas que buscan solucionar el diseño de mallas curriculares balanceadas. El objetivo principal es proveer de un mecanismo que permita simplificar la inspección y análisis de la gran cantidad de datos asociados a BACP valiéndose de una comunicación via imágenes.

6.1. Motivación

La razón principal de la propuesta y construcción de esta herramienta es el hecho de que asociado al diseño de guías para construir procesos heurísticos que tengan por finalidad solucionar BACP, existe una dificultad intrínseca en la lectura y análisis de los datos. Tanto en la generación de una solución inicial, como en la definición de movimientos, validación del cumplimiento de restricciones o medición de la calidad de las soluciones. Siempre es necesario interpretar una serie de descriptores que juntos permiten saber si las decisiones tomadas en el diseño están teniendo los efectos deseados, y a partir de esto determinar qué decisiones futuras serán tomadas. En el caso específico de BACP, resulta de gran importancia conocer el efecto que tienen las estrategias de selección de movimientos y cómo repercute el cambio en la evolución de la malla curricular al elegir distintas funciones de evaluación, factores que con la visualización intuitiva y agrupamiento de los datos son posibles de comprender de manera más fácil y directa. Un ejemplo de esto puede verse en la construcción de la solución factible mencionada en el capítulo anterior.

Llegado cierto punto se hace necesario insertar cursos en periodos que no tienen cursos asignados, para lo cual se toman cursos de periodos que tienen gran carga académica y se insertan en aquellos períodos con problemas para satisfacer las restricciones de mínimo y máximo de créditos y cursos. La labor de verificar el funcionamiento de esta sección del algoritmo junto con verificar cómo evoluciona éste en el cumplimiento de las restricciones es una tarea bastante engorrosa, mientras que observando las imágenes

generadas por la herramienta propuesta pueden usarse varios elementos que simplifican el análisis. En la Figura 6.1 y la Figura 6.2 se muestran algunas de las imágenes señaladas.

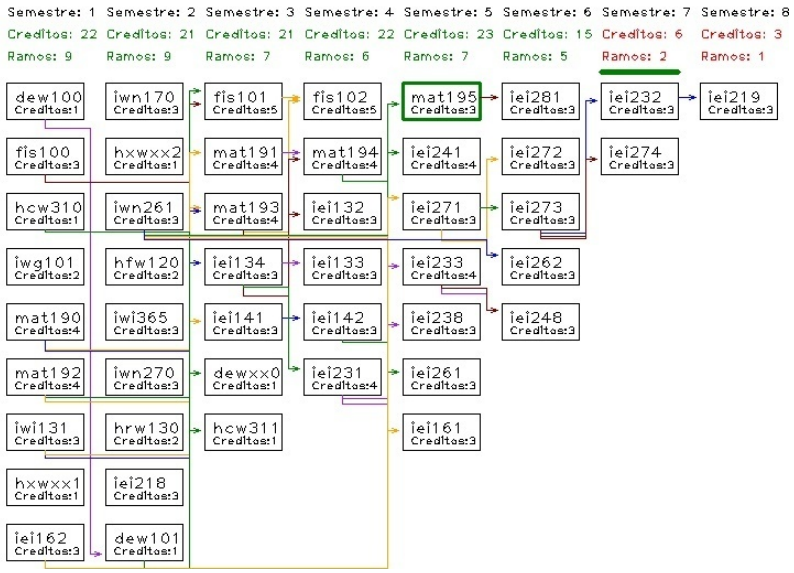


Figura 6.1: Elección de un curso en la generación de la solución inicial. Fuente: Elaboración propia.

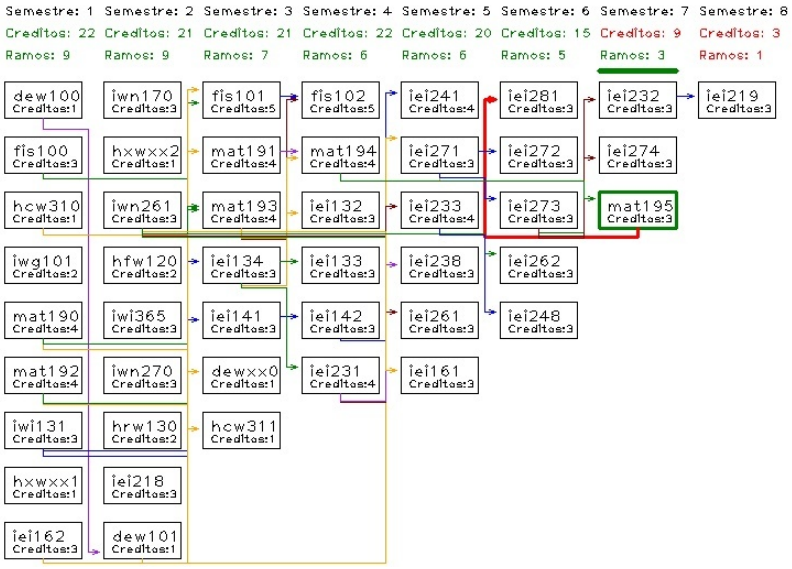


Figura 6.2: Satisfacción de restricciones en la generación de la solución inicial. Fuente: Elaboración propia.

En la Figura 6.1, los periodos 7 y 8 tienen problemas de satisfacción de mínimo de créditos y cursos, las letras en rojo debajo de cada periodo proporcionan una manera rápida de chequear este problema; por otro lado la malla completa no posee problemas de incumplimiento de prerrequisitos, puesto que todas las flechas que los simbolizan están apuntando hacia periodos posteriores (que se encuentran hacia la derecha). A continuación, es posible observar que el curso mat195 de tres créditos que está en el periodo 5 en la primera figura, ha sido seleccionado para ser insertado en el periodo 7, cuyo cambio se refleja en la Figura 6.2. En la Figura 6.2 se observa que la inserción escogida no es posible, pues si bien el mínimo de cursos del periodo 7 ahora está en verde (i.e aquella restricción queda satisfecha), se puede ver una flecha roja apuntando hacia la izquierda que muestra que se está violando una restricción de prerrequisitos.

6.2. Herramienta propuesta

La herramienta propuesta está basada principalmente en las imágenes de las mallas curriculares que presentan las universidades y la simplicidad que presentan para comprender su organización. Así, la estrategia principal es encontrar una forma intuitiva, rápida y fácil de analizar la ejecución del algoritmo, teniendo en cuenta que a medida que éste avanza va generando mallas repetidamente, chequeando el cumplimiento de las restricciones definidas y estableciendo la dirección de avance en base a valores de funciones. En cada paso de la metaheurística se muestra la malla completa con cada periodo, los cursos por periodo, los créditos por periodo y las relaciones de prerrequisitos. En conjunto con esto, se presentan todos los valores de todas las variables asociadas al problema, además de destacar con rojo aquellos valores de restricciones o variables que no están siendo satisfechos y en verde aquellos valores de variables que si lo están.

A partir de lo anterior, una secuencia de imágenes generadas por esta herramienta permite ver la variación en el tiempo de las variables e interrelaciones del problema, permitiendo observar más claramente conflictos de estancamiento del algoritmo y factores importantes a considerar, como por ejemplo cursos que son prerrequisitos de muchos otros o cómo varían las distintas funciones de interés en cada iteración.

6.2.1. Opciones de visualización de malla curricular

Si bien las opciones de uso de la herramienta pueden ser muchas, se proponen las siguientes:

1. **Solución inicial:** esta opción permite ver la solución inicial y cuales son sus variables de estado antes de comenzar la primera iteración.

2. **Solución inicial paso a paso:** esta opción permite ver la generación de la solución inicial paso a paso, cómo fueron asignándose los cursos a los distintos periodos y cuáles fueron las operaciones que el algoritmo Greedy realizó para obtenerla.
3. **Mostrar iteraciones:** esta opción permite ver cuál es el estado de los datos en cada iteración, además del movimiento escogido para la siguiente.

6.2.2. Secciones de la visualización

La Figura 6.3 presenta las secciones de visualización de datos de la herramienta de diseño.

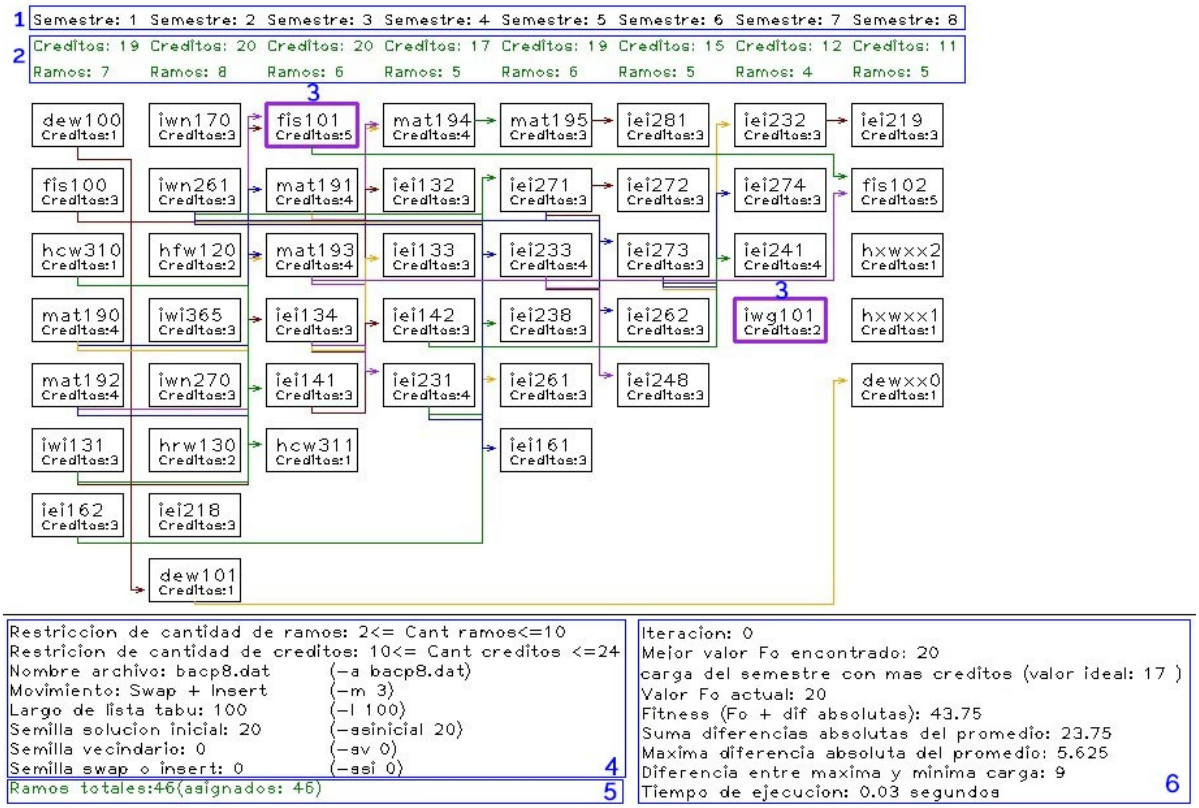


Figura 6.3: Partes de la herramienta de visualización de datos, ejemplo para la instancia BACP8. Elaboración propia.

- **Visualización de períodos:** En la parte superior de la Figura 6.3 destacado con el número 1.

- **Visualización de cursos asignados:** Pueden verse en forma de cajas con la sigla y cantidad de créditos asociados al curso agrupadas en forma vertical debajo del periodo al que están asignados.
- **Visualización de restricciones:** Pueden observarse todas las restricciones asociadas a BACP, las cuales aparecen en verde en caso de ser satisfechas o en rojo en caso contrario. En la Figura 6.3, destacado con 2 se muestran los créditos y cantidad de cursos asignadas a cada periodo. Más abajo, las flechas que salen de un curso y entran a otro representan los distintos prerrequisitos asociados a la instancia. En la parte inferior de la Figura 6.3, destacada con 5 puede apreciarse un indicador que muestra la cantidad de cursos totales de la instancia versus la cantidad de cursos asignados.
- **Visualización de movimientos:** En la Figura 6.3, destacado con 3 los cursos con marco morado representan la parte swap del movimiento, en caso de representar la parte insert se vería destacada sólo un curso y el periodo elegido para hacer la inserción.
- **Visualización de datos del programa:** Destacado con 4 en la Figura 6.3, estos datos proveen información sobre los parámetros utilizados para ejecutar el algoritmo, lo que permite recordar fácilmente cuales son los valores para repetir los experimentos en caso de ser necesario.
- **Visualización de variables y funciones de interés:** Destacado con 6 en la Figura 6.3, se presentan todos aquellos datos que junto al esquema de la malla sirven para hacer un análisis de cómo las decisiones repercuten en la generación del currículum y cómo el algoritmo está guiando la búsqueda. Además de las funciones objetivo y de evaluación, se muestran otros descriptores de estado que servirán para tomar decisiones de diseño. Por ejemplo, observando la variación entre distintas imagenes producidas por la herramienta, podría notarse cuan sensibles son estos parámetros a mejorar en la función objetivo, y como estos pueden ayudar a incorporar un cambio variable de la función de evaluación a lo largo de la ejecución.

6.3. Limitaciones de la herramienta

Las principales limitaciones de la herramienta son de espacio visual, es decir, el tamaño en pixeles de las imágenes que serán mostradas y la cantidad de datos que podrán mostrarse en ellas. Por ejemplo, una instancia BACP12 con 12 periodos y 66 cursos contiene ya un tamaño suficientemente grande, de tal manera que al ser dibujada en pantalla ocupará un espacio considerablemente alto para una resolución 1336x720 pixeles. Así, si bien las instancias más grandes pueden ser dibujadas, se requerirán imágenes cada vez más grandes, lo que dificultará o ralentizará su visualización y

entendimiento al tener que realizar zoom repetidas veces para poder observar distintos sectores de la imagen.

Otro factor a considerar son los movimientos y como estos van afectando al largo de la lista Tabu. Para largos de lista muy grandes, a medida que se van realizando más iteraciones la lista va creciendo cada vez más y se vuelve demasiado grande como para ser incluida en la imagen sin afectar su visualización. Por este motivo se decidió no mostrar la lista Tabu, ya que además de no presentar mayores ventajas respecto de la visualización y no proporcionar información demasiado relevante para el análisis de los datos, ocuparía demasiado espacio de dibujo dentro del área de la imagen.

6.4. Conclusiones

En este capítulo se propone una herramienta cuyo objetivo es facilitar tanto el análisis de los resultados así como el diseño de solvers que buscan resolver problemas BACP. Se trata de una herramienta gráfica que permite visualizar tanto la carga académica como la violación de restricciones y observar un conjunto de datos de interés para un diseñador.

Esta herramienta tiene mucho potencial ya que puede servir de ayuda a cualquier técnica, es decir, no está asociada al Tabu Search que se presenta en esta memoria, lo que la hace versátil y de gran aplicación.

7 Conclusiones

En esta memoria se ha descrito en detalle el BACP el cual es fuertemente NP-Completo. Se presentaron los mayores avances del problema y las diferentes técnicas de resolución en la literatura.

Inicialmente se realizó un recorrido general por los orígenes de BACP, su definición y avances más significativos. Los avances de BACP desde su creación han hecho que logren solucionarse las instancias iniciales propuestas en tiempos pequeños, dando lugar a la generación de nuevas instancias e incluso la proposición de un problema generalizado más complejo de resolver llamado GBACP. A partir de los métodos de resolución vistos en la literatura, se puede concluir que existe una fuerte influencia y en general una tendencia hacia la utilización de modelos de programación entera y satisfacción de restricciones, los cuales, muestran en general los mejores resultados para las instancias inicialmente propuestas.

En base a la metaheurística basada en Tabú Search propuesta y los resultados obtenidos en las diferentes instancias se puede concluir que ésta permite resolver las principales instancias de BACP rápidamente, con alta eficacia y eficientemente. Destaca el hecho de que su eficacia sea cercana al cien por ciento en todas las pruebas realizadas. Lo anterior es principalmente gracias a las estrategias de movimientos elegidos para su exploración y explotación junto con la facultad inherente de Tabú Search para evitar ciclos recordando movimientos realizados. Además, se concluye que el criterio $Min\{max\}$ (minimizar la máxima carga académica de todos los semestres) si bien sirve para probar la optimalidad de una solución, no es un criterio útil a la hora de discriminarlas y guiar al algoritmo. Por esta razón, se propuso un grado más fino para discernir como la desviación media propuesta en el capítulo 4, la cual permite discriminar de mejor manera la calidad de las soluciones. A partir de la resolución de las instancias de CSP Lib se logró llegar a valores equiparables a los mejores resultados de la literatura. Por otro lado, a partir de la resolución de las 100 instancias obtenidas de Bacp12, se corroboró que el algoritmo encuentra soluciones rápidamente y con una alta efectividad. En conjunto con lo anterior, se destaca ampliamente el algoritmo Greedy que genera las soluciones iniciales, el cual permite partir la búsqueda desde diversas soluciones con grados aceptables de calidad. Además se destaca que la metaheurística trabaja sólo con soluciones factibles por lo que los movimientos y entornos de vecindarios están especialmente diseñados para conseguir este efecto.

A partir de la herramienta de visualización propuesta en esta memoria puede concluir-

se que ésta representa una herramienta potencialmente útil para facilitar el análisis de resultados cuando se diseñan algoritmos para resolver BACP, permitiendo tanto hacer *debug* de los algoritmos al poder mostrar paso por paso las soluciones que se generan, como observar si las restricciones están cumplidas y analizar cómo va evolucionando el comportamiento del algoritmo. Aun más, se destaca que la herramienta puede servir de ayuda a cualquier técnica, incluso si ésta no está asociada al Tabú Search, esta última característica la hace versátil y de un impacto potencial grande resultando interesante la creación y comparación de otras metaheurísticas con la ayuda de su uso. Otra característica interesante es que permite la grabación de video con las iteraciones del algoritmo, característica que permite a los diseñadores detectar elementos importantes mientras hacen las pruebas, tales como detectar ciclos o convergencia prematura. Otro elemento destacable es que también puede modificarse para incorporar elementos más complejos, como son las relaciones de dominancia, las cuales permiten establecer cadenas de prerrequisitos y seleccionar inteligentemente los cursos para salir de estancamientos.

Finalmente, a partir del trabajo de investigación, los algoritmos y herramientas propuestos y los resultados obtenidos, es posible dar lineamientos para desarrollo futuro que consideren lo actualmente realizado. Entre estos se considera la implementación del generador de instancias mencionado en la literatura, que permita generar problemas más complejos y con ello testear los límites del algoritmo. Otro es la exploración de nuevas formas para generar los vecindarios como son el uso de las reglas de dominancia. Por último, el desafío más próximo es la resolución de GBACP. Para esto es preciso considerar que este problema se compone en esencia de dos aspectos fundamentales, la resolución de varias mallas curriculares que pueden compartir varios cursos entre sí, y la incorporación de preferencias de profesores que señalan que en determinados periodos no podrán asignarse cursos específicos. A partir de esto, los buenos resultados obtenidos en esta memoria sugieren un buen punto de partida para solucionar GBACP. Más específicamente, teniendo esquema de resolución basado en la generación de una solución inicial factible, una estructura de vecindarios generada a partir de dos movimientos complementarios que tienen sólo soluciones factibles y una función de evaluación como la propuesta, se podría solucionar cada una de las mallas curriculares de una instancia GBACP. Adicionalmente, dado que para cada instancia es necesario resolver varias mallas curriculares, resulta de gran interés considerar la paralelización del algoritmo. Por otro lado, será necesario establecer si las restricciones de preferencias de profesores se considerarán como restricciones blandas o duras, lo cual sería evaluado mediante instancias de prueba junto con el apoyo de la herramienta de visualización. En general es destacable que tanto el análisis de las distintas opciones de tratamiento de restricciones, como la evaluación, diseño y adaptación general de la metaheurística aquí propuesta al GBACP serán hechos utilizando la herramienta de visualización adaptada a las condiciones del nuevo problema.

Bibliografía

[1] Carlos Castro and Sebastian Manzano. Variable and value ordering when solving balanced academic curriculum problems. *Proceedings of 6th Workshop of the ERCIM WG on Constraints*, June 2001.

[2] Marco Chiarandini, Luca Di Gaspero, Stefano Gualandi, and Andrea Schaerf. The balanced academic curriculum problem revisited. *Journal of Heuristics*, 18:119–148, 2012. 10.1007/s10732-011-9158-2.

[3] L. Di Gaspero and A. Schaerf. Hybrid local search techniques for the generalized balanced academic curriculum problem. *Hybrid Metaheuristics*, pages 146–157, 2008.

[4] B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Hybrid modelling for robust solving. *Annals of Operations Research*, 130(1):19–39, 2004.

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

[6] T. Lambert, C. Castro, E. Monfroy, and F. Saubion. Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. *Artificial Intelligence and Soft Computing - ICAISC 2006*, 4029:410–419, 2006.

[7] Belén Melián, José A Moreno Pérez, and J Marcos Moreno Vega. Metaheurısticas: una visión global. *Revista Iberoamericana de Inteligencia Artificial*, 19:7–28, 2003.

[8] Jean-Noël Monette, Pierre Schaus, Stéphane Zampelli, Yves Deville, and Pierre Dupont. A cp approach to the balanced academic curriculum problem. In *Symcon*, volume 7, pages 95–99, 2007.

[9] 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.

[10] P. Schaus, Y. Deville, P. Dupont, and J.C. Régin. Simplification and extension of the spread constraint. In *Third international workshop on constraint propagation and implementation*, pages 77–91, 2006.

[11] Pierre Schaus et al. *Solving balancing and bin-packing problems with constraint programming*. PhD thesis, Ecole polytechnique de Louvain D'epartement d'Ingénierie Informatique Université catholique de Louvain Louvain-la-Neuve, Belgium, August 2009.

[12] Scheduling and Timetabling Research Group at the University of Udine. The generalized balanced academic curriculum problem. *http://satt.diegm.uniud.it/index.php?page=gbac*, June 2012.

[13] J.A.A. Solis. *Un modelo basado en optimizaciön para balancear planes de estudio en Instituciones de Educaciön Superior*. PhD thesis, UPAEP,Puebla, 2008.

[14] J.A.A. Solís, J.L.M. Flores, M.C. Ríos, and J.P.N. de la Parra. El problema de balancear un plan de estudios: Un modelo matemático. *Segundo Taller Latino Iberoamericano de Investigación de Operaciones, México*, 2007.

[15] J.A.A. Solis, P. Puebla, and M.J.L.M. Flores. Estudio de diferentes conceptos de balance en modelos bacp. *Tercer Taller Latino Iberoamericano de Investigación de Operaciones, México*, 2009.

[16] C. Vergara. Desarrollo de un modelo de programación lineal para la generación de mallas curriculares balanceadas. *Memoria de Ingeniero de Ejecución en Informática, UTFSM, Valparaıso, Chile*, 1994.