Universidad de Cienfuegos "Carlos Rafael Rodríguez"

Facultad de Ingeniería

Tesis de Maestría

para optar por el título de Máster en Matemática Aplicada

Modelación Matemática y Solución del problema de Balance de Carga Docente en la Universidad de Cienfuegos

Autor: **Boris Pérez Cañedo**

Tutores: **MSc. Ridelio Miranda Pérez**
MSc. Miguel Santana Justiz

2014

Dedicatoria

A Anabelia, mi hija, a quien amo infinitamente y que continuamente me sorprende con su inteligencia.

A Ismaray, mi esposa, espero que tanto esfuerzo nos sea recompensado. Te amo.

Agradecimiento

A mis amigos y tutores, MSc. Miguel Santana Justiz y MSc. Ridelio Miranda Pérez por sus contribuciones durante el análisis del problema y principalmente por la amistad y el apoyo brindado.

Al MSc. Pedro Roberto Suárez Surí por su ayuda con las pruebas estadísticas utilizadas.

Al DrC. Domingo Curbeira Hernández por su ayuda invaluable en la revisión de este trabajo.

Resumen

En este trabajo se formula un modelo matemático heurístico general para la modelación del problema de balance de carga docente en la Universidad de Cienfuegos. El modelo obtenido no puede ser resuelto por los algoritmos clásicos de la programación lineal. De modo que se propone un algoritmo de solución basado en Búsqueda Tabú. El algoritmo propuesto implementa una metodología que promueve la convergencia hacia un punto factible (posiblemente óptimo) del problema. El análisis de los resultados numéricos del modelo y el algoritmo propuestos, expuesto en el capítulo tres, confirma que en efecto la convergencia global está prácticamente garantizada para el caso que se consideren turnos de clase de duración constante. Existe además evidencia numérica para suponer que el algoritmo propuesto conserva la propiedad de convergencia global cuando se consideran turnos de clase de duración arbitraria. Como resultado práctico se tiene el desarrollo de una aplicación de escritorio (QBalance), en la que se codifican el modelo y el algoritmo propuestos, que asiste a los planificadores docentes en la confección de balances de carga. La utilización de esta aplicación permite obtener balances de carga docente que poseen un mejor equilibrio de horas clases semanales respecto a aquellos confeccionados manualmente por el personal de planificación docente, reduce a cero los errores del balance de carga docente obtenido y disminuye significativamente (de dos días a aproximadamente 40 segundos) el tiempo de planificación de la carga docente de un grupo.

Índice general

Índice de figuras

Índice de Tablas

Índice de códigos en Python

Índice de algoritmos en pseudocódigo

Introducción

La planificación docente (desde el punto de vista administrativo) está conformada por un conjunto de orientaciones generales y particulares para garantizar el éxito del período docente; estas orientaciones incluyen: indicaciones sobre la fecha de inicio del período docente y las semanas que lo integran; disponibilidad de aulas, laboratorios, recursos tecnológicos; calendario de actividades extracurriculares, días feriados; asignación de responsabilidades en el proceso; indicaciones a los decanos, jefes de carreras y profesores en general; descripciones de las asignaturas que integran cada período docente con sus respectivos fondos de tiempo; modelos de plan de impartición y plan calendario (P1) para las asignaturas; disposiciones sobre la realización de los cortes evaluativos, los exámenes finales, extraordinarios y especiales, entre otras.

La automatización de la gestión universitaria constituye para las universidades cubanas uno de sus principales retos. En este sentido los autores en (Lao y Gómez, 2007) plantean: "... la utilización de las Tecnologías de la Información y las Comunicaciones en la Gestión Académica, posibilita la integración de recursos, sujetos e instituciones para resolver problemas comunes, lo cual contribuye al perfeccionamiento del Proceso Docente Educativo y por consiguiente, a ponerlo a tono con las actuales exigencias de la formación de estudiantes universitarios en la actual sociedad de la información y el conocimiento ...". En particular, la planificación docente es sin lugar a dudas determinante para el éxito del proceso docente educativo, es considerada por los autores en (Tamayo, Campaña, y Expósito, 2007) como: "... la actividad encargada de organizar adecuadamente el proceso docente educativo para cumplir la función sustantiva de la universidad (...) la formación del profesional que la sociedad exige". La gestión automatizada de esta actividad contribuye significativamente a este fin.

Dos tareas esenciales de la planificación docente son, la confección de los balances de carga docente y la confección de los horarios docentes (P4). El balance de carga docente se realiza al inicio de cada semestre cuando el departamento de planificación docente define los requisitos docentes, según los planes de estudio[1] (PE) de las carreras, el fondo de tiempo declarado para el semestre y el número de asignaturas y semanas que lo componen para cada curso de cada facultad. Las facultades solicitan a los departamentos las asignaturas a impartir según los PE; los jefes de departamento asignan los profesores a las asignaturas y estos últimos diseñan los programas de las asignaturas y los P1 de acuerdo con las orientaciones dadas en los PE. Cada P1 contiene el fondo de tiempo, la cantidad de turnos de clase, la cantidad de horas por turno de clase y el tipo de clase. El departamento de planificación docente recibe los P1 de las asignaturas y luego de confirmar que cumplen con los requisitos establecidos proceden a confeccionar los balances de carga docente de cada grupo independientemente. Para confeccionar un balance de carga docente se tienen en cuenta los elementos siguientes:

Para cada asignatura:

- Fondo de tiempo de la asignatura.
- Cantidad de turnos de clase planificados por el profesor en el P1.
- Duración de cada turno de clase.

Para cada semana del período:

- Restricciones de cantidad de horas disponibles para una semana, como resultado de: días feriados, eventos culturales, la etapa de pruebas de ingreso, otras actividades que afectan la docencia.

[1] Véanse los artículos 21, 22 y 23 de la resolución No. 210/07 del Ministerio de Educación Superior de la República de Cuba.

Los planificadores docentes de la Universidad de Cienfuegos elaboran una plantilla consistente en un arreglo bidimensional de asignaturas por las columnas y semanas por las filas. Proceden ubicando los turnos de cada asignatura, en función del fondo de tiempo correspondiente en cada semana. Balancear la carga docente consiste en modificar cada semana, asignando o eliminando turnos de clase de las asignaturas, hasta lograr un equilibrio de la carga docente; esto es, se pretende que las horas de clase semanales se mantengan aproximadamente iguales durante todas las semanas del semestre. Esta actividad la realizan los planificadores docentes de forma manual sobre la base de la experiencia. Los inconvenientes que presenta este proceder están dados por el tiempo excesivo (usualmente dos o tres días) que toma obtener el balance de un grupo, lo engorroso del procedimiento ocasiona errores y como consecuencia atrasos en la planificación, la imposibilidad de evaluar otras alternativas, la dificultad para responder adecuadamente ante situaciones imprevistas en la planificación y la obtención de balances de carga muy lejos de poder llamarse óptimos. Esta situación conduce al planteamiento del siguiente problema científico.

Problema científico:

¿Cómo planificar la carga docente para cumplir con los requisitos del departamento de planificación docente de la Universidad de Cienfuegos?

Hipótesis:

Si se presenta un modelo matemático heurístico para resolver el problema de balance de carga docente entonces se disminuirá el tiempo de confección del mismo, no se presentarán errores y se obtendrá un mejor equilibrio de horas clases semanales durante el período de planificación.

- **Variable independiente**: Modelo matemático heurístico.
- **Variables dependientes**: Tiempo de planificación, Errores del balance de carga docente y Evaluación de la función objetivo del modelo heurístico (mide el equilibrio de horas clases semanales durante el período de planificación).

Objeto de estudio:

El proceso de planificación docente.

Campo de Acción:

El balance de carga docente en la Universidad de Cienfuegos.

Objetivo general de la investigación:

Modelar matemáticamente el problema de balance de carga docente en la Universidad de Cienfuegos para la automatización de este proceso.

Objetivos específicos:

- Formalizar un modelo matemático heurístico que refleje el proceso de balance de carga actual.
- Implementar un algoritmo de solución para el modelo matemático heurístico.
- Codificar un software para la automatización del proceso de balance de carga docente en la Universidad de Cienfuegos.
- Validar numéricamente el modelo matemático heurístico y el algoritmo de solución propuesto.

Tareas de la investigación:

- Valoración crítica de antecedentes y resultados relacionados con el tema.
- Valoración crítica de algoritmos de solución que permitan resolver el modelo matemático.
- Formulación de un modelo matemático que cumpla con los requisitos identificados para la solución del problema de balance de carga docente en la Universidad de Cienfuegos.
- Selección de un algoritmo adecuado para la solución del modelo matemático.

- Selección de un lenguaje de programación para la automatización del balance de carga docente.
- Valoración de la convergencia hacia el óptimo global del algoritmo seleccionado.
- Contrastación y valoración crítica de los resultados del modelo matemático y las soluciones empíricas.

Metodología

De los métodos Teóricos se utilizan:

- El Analítico-Sintético para resumir y sintetizar la información recopilada en la literatura relacionada con el objeto de estudio.
- El Histórico-Lógico para la obtención de las relaciones que caracterizan al proceso de balance de carga docente en su decursar histórico.
- El Hipotético-Deductivo para, a partir de la observación del fenómeno objeto de estudio, construir la hipótesis de investigación.
- La Modelación para representar matemáticamente el problema de balance de carga docente y la obtención de un algoritmo de solución.

De los métodos empíricos se utilizan:

- El Análisis de Documentos para la revisión de la información de las fuentes primarias, de la que se tomarán los datos relacionados con el objeto de estudio. Estos documentos son facilitados por el personal del Departamento de Planificación Docente de la Universidad de Cienfuegos.
- La Entrevista para complementar y constatar la información obtenida con los otros métodos de investigación.

Novedad de la investigación:

Formalización de una metodología específica para la solución de problemas que se ajusten al modelo de balance de carga docente propuesto.

Aporte práctico: El desarrollo de un modelo matemático heurístico para la automatización del balance de carga docente en la Universidad de Cienfuegos.

El informe está estructurado en:

- Resumen.
- Introducción.
- Tres Capítulos.
- Conclusiones Generales.
- Recomendaciones.
- Referencias Bibliográficas.
- Anexos.

Capítulo 1: En este capítulo se revisa la literatura especializada que ha tratado la modelación y solución, por métodos, algoritmos y técnicas diversas, de los problemas de confección de balances de carga docente y horarios docentes. Se muestra la relación existente entre los problemas de confección de balances de carga docente y los de confección de horarios docentes. Se revisan además los pasos dados en la automatización de los procesos docentes llevados a cabo en las universidades cubanas, en particular aquellos directamente relacionados con los balances de carga docente y la confección de horarios docentes.

Capítulo 2: En este capítulo se formula el modelo matemático heurístico general para el problema de balance de carga docente en la Universidad de Cienfuegos y se selecciona un algoritmo de solución que utiliza la estructura propia del modelo. Para la obtención de este modelo se realizan simplificaciones del problema, lo que posibilita ir de lo particular a lo general durante la etapa de modelación.

Capítulo 3: En este capítulo se exponen los resultados de experimentos numéricos realizados con los modelos obtenidos en el capítulo dos y se contrastan las soluciones obtenidas con el modelo matemático heurístico general para el balance de carga docente con la soluciones manuales del personal de planificación docente. Se describen brevemente las bibliotecas del lenguaje de programación Python que son utilizadas para la automatización del algoritmo de solución descrito en el capítulo dos.

Capítulo 1

Marco teórico para la confección de balances de carga y horarios docentes

Introducción

En este capítulo se revisa la literatura especializada que ha tratado la modelación y solución, por métodos, algoritmos y técnicas diversas, de los problemas de confección de balances de carga docente y horarios docentes. Se muestra la relación existente entre ambos problemas y su importancia para el proceso de planificación docente en la Universidad de Cienfuegos. Se revisan además los pasos dados en la automatización de los procesos docentes llevados a cabo en las universidades cubanas, en particular aquellos directamente relacionados con los balances de carga docente y la confección de horarios docentes.

1.1. Modelos matemáticos para la confección de horarios docentes

Los horarios para actividades de diversa naturaleza son conocidos en la literatura como *Timetabling*[1]. En la década de 1960 ocurre una explosión en la literatura que aborda los modelos y algoritmos para la confección de horarios. A partir de ese momento hasta la fecha, puede afirmarse que no se ha dejado de publicar un solo artículo sobre el tema. En particular para las universidades, los problemas de horarios docentes constituyen una de las tareas más difíciles que llevan a cabo los planificadores docentes de estas instituciones. La solución manual de estos problemas requiere un extraordinario dominio del tiempo y los recursos. Elaborar un horario docente con la menor cantidad posible de coincidencias en los horarios de los profesores, aulas, asignaturas, etc. es muy difícil. En general estos problemas, por su naturaleza combinatoria, constituyen problemas NP-duros.[2]

Varios modelos matemáticos y algoritmos de solución han sido propuestos para la

[1]Se adopta con cierta frecuencia esta denominación incluso en español; el autor de este trabajo utiliza el término *horario* en todo el documento.

[2]Una clase o propiedad de los problemas de búsqueda computacionales. Véase la sección 1.5 para más detalles. Consúltese además (Dictionary.com, 2014)

confección de horarios docentes. No obstante, todos estos modelos y algoritmos son estrechamente dependientes de la situación específica que presenta la institución, por lo que la reproducción exacta de los resultados alcanzados en otras instituciones es esencialmente imposible. Aún así, las estrategias generales para el manejo de los distintos tipos de restricciones sobre los recursos pueden, hasta cierto punto, conservarse. Por ejemplo, los autores dividen las restricciones del problema en dos grupos: restricciones duras (restricciones que tienen que ser satisfechas sin excepción y cuyo incumplimiento generaría un horario no válido) y restricciones suaves, cuyo cumplimiento es deseable pero no imprescindible para la obtención de un horario válido. Como restricciones duras pueden tenerse, entre otras: ningún recurso (profesor, grupo de estudiantes y aula) puede ser asignado a distintas actividades en el mismo espacio de tiempo, el aula asignada a una actividad tiene que cumplir los requisitos necesarios para el desarrollo de ésta, es decir, posee la capacidad (asientos disponibles), recursos tecnológicos necesarios para llevar a cabo la actividad, etc.

En (Azadeh, Gholizadeh, y Jeihoonian, 2013) los autores proponen un modelo dinámico multi-objetivo de programación no lineal en enteros mixtos para la solución del problema. El modelo es finalmente resuelto utilizando el software GAMS[3]. En (Perzina y Ramik, 2013) los autores plantean una modelación con variable de decisión binaria y restricciones suaves difusas para capturar la incertidumbre presente en los datos.

En Cuba se ha incursionado en el tema destacándose, entre otros, un estudio realizado en la Universidad de Holguín (Tamayo y cols., 2007), donde es empleado un algoritmo de satisfacción de restricciones. Uno de los proyectos más importantes para la informatización de los procesos de las universidades cubanas es el Sistema de Gestión de la Nueva Universidad (SIGENU). El proyecto SIGENU surge en Junio de 2004 a solicitud de la dirección del Ministerio de Educación Superior de Cuba (MES), como requisito a las necesidades de automatización de los procesos fundamentales de la gestión académica de una Institución de Educación Superior (IES) en todas sus modalidades de estudio. El sistema está compuesto por varios módulos que gestionan la información de un estudiante desde que se matrícula hasta que se gradúa o causa baja definitiva. En el curso 2008-2009 fue implantado en el Instituto Superior de Relaciones Internacionales (ISRI) y el Instituto Superior de Diseño Industrial (ISDI).(de desarrollo SIGENU-CUJAE, 2010)

Dentro de las tareas de planificación docente objetos de automatización se encuentran, la

[3]Sistema de modelado algebraico general (GAMS). Es un sistema de modelado de alto nivel para la programación matemática y la optimización.

planificación de la carga docente y la confección de los horarios docentes. A la Universidad de Cienfuegos le fue dada en 2008 la tarea de la automatización de los horarios docentes. En este proyecto se persigue además, lograr una uniformidad en la confección de horarios docentes con el menor impacto posible en la forma en que realizan esta actividad los planificadores de las universidades cubanas. En este sentido, (Cartaya, 2009, 2013) propone un modelo matemático lineal multi-objetivo con variable de decisión binaria para la solución del problema en la Universidad de Cienfuegos. El autor utiliza el enfoque de programación por metas (Charnes, Cooper, DeVoe, Learner, y Reinecke, 1968) y el método de jerarquías analíticas (Saaty, 1980) para la determinación de los pesos de las metas. Aunque se han logrado avances en el tema de la confección de horarios docentes, la automatización de los balances de carga docente constituye aun un problema no resuelto que presenta un vínculo estrecho con el problema de horarios docentes. La siguiente sección trata precisamente la relación entre ambos problemas.

Luego de una extensa búsqueda bibliográfica, el autor de este trabajo no ha encontrado en la literatura consultada referencia alguna a la modelación matemática del problema de balance de carga docente. En (Tamayo y cols., 2007) se tiene entre los supuestos planteados para la confección de un horario docente, "...la existencia de un balance de carga docente normalizado..."[4], pero no se especifica si este balance es realizado manual o de manera automatizada, y no se presenta ningún modelo matemático para este problema. El resto de las referencias a este tipo de problema son verbales, obtenidas durante presentaciones en eventos científicos y entrevistas con el personal de planificación docente de la Universidad de Cienfuegos.

No obstante, en el contexto de la automatización de actividades de planificación docente, se encuentra el problema de balance curricular propuesto inicialmente por Carlos Castro en (Castro y Manzano, 2001) y revisitado por el autor y otros en (Lambert, Castro, Monfroy, y Saubion, 2006). El problema de balance curricular consiste básicamente en la ubicación de asignaturas en períodos docentes, respetando el orden de precedencia de las asignaturas y minimizando el balance curricular en término de los créditos aportados por cada una de las asignaturas. El autor de este trabajo revisa este problema en la sección 1.2. Este problema es, de los revisados en la bibliografía consultada, el que más se asemeja con el problema de balance de carga docente, tanto en el contexto como en su formulación

[4]Para los autores del artículo citado, normalizado significa: "equilibrio de clases para un grupo de estudiantes en un período"; el autor de este trabajo considera que realmente se intentó decir: equilibrio en las horas de clase semanales para un grupo de estudiantes en un período.

matemática.

Si el problema de horario docente consiste en asignar lo mejor posible las clases para los estudiantes y profesores en bloques de tiempo semanales, en determinadas aulas, bajo ciertas restricciones y requiere el conocimiento previo de la cantidad de turnos de clase de cada asignatura que serán impartidos en cada semana, entonces el problema de balance de carga docente consiste precisamente en determinar la cantidad de turnos de cada asignatura en cada semana para garantizar un equilibrio en la carga docente durante todo el semestre. El autor de este trabajo infiere entonces que la solución dada al problema de balance de carga docente puede ser empleada como solución inicial para el problema de horario docente. Aunque no será demostrado, también se presupone que el problema de balance de carga es un problema NP-duro. Los resultados de los experimentos numéricos expuestos en el capítulo tres apoyan esta afirmación.

El problema de balance de carga docente es también una de las tareas más difíciles que realiza el personal de planificación docente. La Universidad de Cienfuegos oferta dieciséis carreras. Los planificadores docentes deben planificar la carga docente de cada grupo de estas carreras, tomando a cada planificador aproximadamente dos días la culminación de un solo balance. El balance de carga docente de un semestre se completa en aproximadamente tres meses de trabajo y esto es solo para comenzar con la planificación de los horarios docentes.

Cuando se confeccionan balances de carga docente mediante un modelo y/o procedimiento, ya sea manual o automatizado, tienen que procurarse los siguientes requisitos:

- No modificar el fondo de tiempo de la asignatura.
- Respetar la cantidad de turnos de clase planificados por el profesor en el P1.
- Respetar la duración de cada turno de clase, tal y como fue especificado en el P1.
- Sea flexible, en el sentido de que permita a los planificadores bloquear semanas (no planificarlas), establecer fondos de tiempo para cada semana independientemente (esto permite considerar situaciones tales como: días feriados, eventos culturales, la etapa de pruebas de ingreso y otras actividades que afecten la docencia.) y que permita cierto grado de negociación entre los planificadores y los profesores con respecto a la cantidad de turnos de clase a impartir de sus asignaturas en cada semana del semestre.

El problema de balance de carga docente constituye también un problema de asignación. La siguiente sección aborda dos de los problemas clásicos de asignación con el propósito de encontrar posibles similitudes con el problema estudiado.

1.2. El problema de balance de carga docente y otros problemas de asignación

Balancear la carga docente significa *asignar* una cantidad de turnos de clases de cada asignatura a cada semana del semestre, de forma tal que las horas clases semanales se mantengan aproximadamente iguales durante las semanas del semestre. Conviene entonces revisar algunos problemas de asignación que han sido tratados en la literatura y que pudieran revelar algunas similitudes con el problema en cuestión.

1.2.1. El problema de balance curricular

De acuerdo con (Castro y Manzano, 2001), " ...un factor clave a tener en cuenta al evaluar el éxito académico de los estudiantes es la carga académica que tienen en cada período académico...". Para medir esta carga se asigna a cada asignatura un número de créditos que básicamente representan el esfuerzo requerido para seguir con éxito la asignatura. La carga académica de cada período es calculada sumando los créditos de cada asignatura impartida durante el período. Además, se imponen algunas restricciones específicas en la confección de un plan de estudios. Estas restricciones contemplan, entre otras, la carga máxima permitida por período con el objetivo de evitar la sobrecarga, junto a algunas relaciones de precedencia entre las asignaturas. Se parte del supuesto de que una carga en equilibrio promueve el éxito de los estudiantes.

El problema de balance curricular consiste en la asignación de asignaturas a períodos de forma tal que la carga académica de cada período será equilibrada, es decir, lo más similar posible. La formulación matemática de este problema tomada directamente del artículo citado es la siguiente:

Sean m, n, α_i, β, γ, δ y ϵ, el número de asignaturas, períodos, número de créditos de la asignatura i, mínima carga académica permitida por período, máxima carga académica permitida por período, mínima cantidad de asignaturas por período y la máxima cantidad de asignaturas por período respectivamente.

$$x_{ij} = \begin{cases} 1, \text{ si la asignatura } i \text{ se imparte en el período } j \\ 0, \text{ en otro caso} \end{cases} \tag{1.1}$$

c_j: carga académica del período j.

c: máxima carga académica en todos los períodos.

Función objetivo

$$\text{mín}\, c = \text{máx}\{c_1,\, c_2, \ldots,\, c_n\} \tag{1.2}$$

Restricciones

- La carga académica del período j está definido por:

$$c_j = \sum_{i=1}^{m} \alpha_i \cdot x_{ij} \quad \forall j = 1, \ldots, n \tag{1.3}$$

- Todas las asignaturas i deben ser asignadas a algún período j.

$$\sum_{j=1}^{n} x_{ij} \quad \forall i = 1, \ldots, m \tag{1.4}$$

- La asignatura b tiene a la asignatura a como prerrequisito.

$$x_{bj} \leq \sum_{r=1}^{j-1} x_{ar} \quad \forall j = 2, \ldots, n \tag{1.5}$$

- La carga académica máxima esta definida por $c = \text{máx}\{c_1,\, c_2, \ldots,\, c_n\}$. Esto puede ser representado por el conjunto de restricciones lineales siguientes:

$$c_j \leq c \quad \forall j = 1, \ldots, n \tag{1.6}$$

- La carga académica del período j debe ser mayor o igual que el mínimo requerido.

$$c_j \geq \beta \quad \forall j = 1, \ldots, n \tag{1.7}$$

- La carga académica del período j debe ser menor o igual que el máximo permitido.

$$c_j \leq \gamma \quad \forall j = 1, \ldots, n \tag{1.8}$$

- El número de asignaturas a impartir en el período j tiene que ser mayor o igual que el mínimo permitido.

$$\sum_{i=1}^{m} x_{ij} \geq \delta \quad \forall j = 1, \ldots, n \tag{1.9}$$

- El número de asignaturas a impartir en el período j tiene que ser menor o igual que el máximo permitido.

$$\sum_{i=1}^{m} x_{ij} \leq \epsilon \quad \forall j = 1, \ldots, n \tag{1.10}$$

Los autores consideraron en la experimentación tres carreras con 8, 10 y 12 períodos respectivamente. Los autores intentaron resolver los problemas utilizando el software lp_solve[5]. Los resultados muestran que el problema con 8 períodos fue resuelto por lp_solve en 1460 segundos. En el caso del problema con 10 períodos, lp_solve no proporcionó la solución óptima luego de haber transcurrido 5 horas en ejecución. Por último, en el caso del problema con 12 períodos lp_solve no obtuvo la solución óptima luego de haber transcurrido un 1 día en ejecución. Estos resultados muestran que la utilización de las técnicas de programación lineal en enteros no manejan eficientemente instancias grandes de problemas combinatorios como el de balance curricular. En este sentido los autores implementaron una técnica conocida por programación de restricciones. La técnica de programación de restricciones se ocupa de problemas de optimización utilizando la misma idea básica de la verificación de la satisfacción de un conjunto de restricciones. Si se está tratando con un problema de minimización, la idea es utilizar una cota superior que representa la mejor solución posible obtenida hasta ahora. Entonces se resuelve una secuencia de problemas de satisfacción de restricciones cada uno dando una mejor solución con respecto a la función de optimización. El empleo de esta técnica permitió a los autores la solución eficiente de estos problemas.

1.2.2. Los problemas de asignación clásicos[6]

Supóngase que n individuos ($i = 1, 2, \cdots, n$) están disponibles para n ($j = 1, 2, \cdots, n$) tareas y que además se tiene una matriz de pesos $R = (r_{ij})$, donde r_{ij} son enteros positivos para toda i y j. Una asignación consiste en la elección de una tarea j_i para cada individuo i de forma que ninguna tarea sea asignada a dos individuos diferentes. El problema de asignación plantea: ¿Para qué asignación es mayor la suma $r_{1j_1} + r_{2j_2} + \cdots + r_{nj_n}$ de los pesos?

Existe un algoritmo combinatorio bien conocido debido a Harold W. Kuhn (El método húngaro) (Kuhn, 1955) que resuelve este problema en tiempo polinomial.

[5] lp_solve, un software para la solución de problemas de programación lineal en enteros.

[6] Basado en parte en el trabajo original de Harold W. Kuhn.

La formulación de este problema como un modelo de programación lineal en variables binarias es la siguiente:

Sea:

$$x_{ij} = \begin{cases} 1, & \text{si la tarea } j \text{ se asigna al individuo } i \\ 0, & \text{en otro caso} \end{cases}$$

Cada tarea es asignada a un solo individuo.

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\} \tag{1.11}$$

Cada individuo realiza una tarea.

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in \{1, 2, \cdots, n\} \tag{1.12}$$

$$\text{máx} \sum_{j=1}^{n} \sum_{i=1}^{n} r_{ij} x_{ij} \tag{1.13}$$

Además del método húngaro pueden aplicarse algoritmos de la programación lineal binaria, aunque sin lugar a dudas el método húngaro lo resuelve mucho más eficientemente.

El siguiente problema, inicialmente formulado en (Koopmans y Beckmann, 1957), también NP-duro (Sahni y Gonzalez, 1976), recuerda el problema anterior, pero la diferencia radica en que ahora la función objetivo es cuadrática. Es conocido en la literatura como el problema de asignación cuadrático.

Se tiene un conjunto de n fábricas y un conjunto de n locaciones. Para cada par de locaciones se conoce la distancia $D = (d_{ih})$ entre ellas, donde , para cada par de fábricas se conoce un peso o flujo $W = (w_{jk})$ (por ejemplo, la cantidad de materiales transportados entre ambas fábricas), se conoce además los costos $C = (c_{ij})$ de asignar (construir) la fábrica i a la locación j , donde $d_{ih} \in \mathbb{R}^+$, $w_{jk} \in \mathbb{R}^+$ y $c_{ij} \in \mathbb{R}^+$. Se desea asignar todas las fábricas a diferentes locaciones con el objetivo de minimizar las distancias multiplicadas por los flujos o pesos correspondientes más los costos de asignación.

Sea:

$$x_{ij} = \begin{cases} 1, & \text{si la fábrica } j \text{ se asigna a la locación } i \\ 0, & \text{en otro caso} \end{cases}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j \in \{1, 2, \cdots, n\}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in \{1, 2, \cdots, n\}$$

$$\text{mín} \sum_{i,j=1}^{n} \sum_{h,k=1}^{n} w_{jk} d_{ih} x_{ij} x_{hk} + \sum_{i,j}^{n} c_{ij} x_{ij}$$

Para la solución de estos tipos de problemas han sido empleados métodos clásicos (Bazaraa y Elshafei, 1979), (de Klerk, Sotirov, y Truetsch, 2013) y meta-heurísticos[7] (Ramkumar y Ponnambalam, 2006),(Tseng, 2006), (Cetin Demirel y Duran Toksari, 2006) y (Tasgetiren, Pan, Suganthan, y Dizbay, 2013). Los métodos clásicos encuentran la solución óptima solo en instancias relativamente pequeñas del problema. La solución empleando métodos meta-heurísticos permite, sin embargo, una exploración más eficiente del espacio de búsqueda; pero generalmente solo proporcionan soluciones suboptimales. Como puede observarse, la solución de estos problemas continúa siendo objeto de investigación en la actualidad.

Existen muchos problemas, sobre todo en problemas de corte económico e industrial, cuya modelación conduce a la solución de un problema de asignación cuadrático. Por ejemplo, en (Geoffrion y Graves, 1976) los autores tratan el problema de secuenciar órdenes de producción de un grupo de productos en líneas de producción similares con el objetivo de minimizar los costos de producción, preparación y ajuste de las líneas de producción. Este modelo fue aplicado en un complejo de reactores químicos en las industrias Dart. El problema consistía en cinco reactores de un tipo que producían veinte diferentes productos mensualmente, mientras que otro reactor produce cerca de seis productos mensuales. Debido a que los productos son producidos en reactores diferentes, el problema de secuenciación se divide en dos problemas separados. Los resultados muestran que la planificación manual resultó un 35 %, 22 % y 25 % más costosa, para las instancias resueltas, que la solución propuesta por estos autores.

Otra aplicación es planteada por Alwalid N. Elshafei en (Elshafei, 1977), esta aplicación consiste en el diseño de un hospital; en esta situación donde muchas vidas están en juego, es importante que el equipo de diseño tome las precauciones necesarias para garantizar que la distribución de los departamentos de la instalación sea la más beneficiosa para los pacientes y los proveedores de atención. Elshafei, del Instituto de Planificación Nacional en El Cairo, Egipto, investigó la asignación óptima de los departamentos específicos o

[7]Ver sección 1.4

clínicas (sala de emergencia, rayos X, etc.) dentro de un hospital. Una asignación óptima según la definición de Elshafei es aquella que minimiza la distancia total recorrida por los pacientes entre las clínicas, medida en pacientes-metros por año. Por ejemplo, no es ninguna sorpresa que la sala de emergencias en casi todos los hospitales del mundo se encuentre ubicada en la parte delantera de la instalación, reduciendo así al mínimo la distancia total a recorrer por un paciente que necesite atención de urgencia. La idea de colocar a la sala de emergencia en cualquier otro lugar es intuitivamente contraproducente e ilógico. En (Elshafei, 1977), el autor se centró específicamente en el creciente problema de la superpoblación del departamento de asistencia ambulatoria en un gran hospital de El Cairo. El departamento se compone de 17 clínicas que tratan a un promedio de 700 pacientes por día. La mala colocación de las clínicas combinada con el volumen cada vez más abrumador de tráfico entre ellos estaba causando retrasos y congestión. El autor formuló el problema anterior como un problema de decisión y utilizó los algoritmos y técnicas de la investigación de operaciones para formar una mejor distribución de los departamentos. La tarea consistía, esencialmente, en asignar n clínicas a n lugares dentro del departamento. Como resultado el autor logró eliminar cerca de un 20 % de tráfico innecesario, resultando en un tratamiento médico de mejor calidad.

En (Burkard y Offermann, 1977) los autores plantean un modelo de asignación cuadrática para mejorar la disposición de las teclas de una máquina de escribir. Los datos lingüísticos necesarios se obtuvieron contando los diferentes pares de letras en textos de más de 100.000 letras. Los teclados mejorados se proporcionan para los idiomas: Inglés, francés, alemán y neerlandés, así como para textos mixtos. Los métodos propuestos por estos autores producen mejoras de un 7 % a un 10 % en comparación con el teclado estándar internacional.

El autor de este trabajo considera que los problemas de asignación clásicos revisados no poseen similitudes destacables con el problema de balance de carga docente que permita la utilización, por ejemplo, del método húngaro. En el caso del problema de asignación cuadrático, su función objetivo es cuadrática al igual que el modelo planteado en la sección 2.14. Una diferencia esencial que posee el problema de balance de carga docente con ambos problemas de asignación es que, por lo general, se precisa asignar más de un turno de clase semanalmente y además a todas las semanas no tiene necesariamente que ser asignado un turno de clase. En el caso del problema de balance curricular, resulta interesante haber encontrado una aplicación directa vinculada estrechamente con el objeto de estudio de esta investigación. El problema de balance curricular guarda una estrecha relación con el

problema de balance de carga docente ya que en ambos se pretende obtener un balance o equilibrio, de créditos en el caso del problema de balance curricular y de horas clase en el caso del problema de balance de carga docente. Una diferencia esencial radica que la asignación en el problema de balance de carga docente es la de un encuentro de una asignatura a una semana determinada, apareciendo de esta forma un tercer índice para la variable de decisión que agrega dificultad al modelo y lo hace diferente al planteado por (Castro y Manzano, 2001) para el balance curricular. Por otro lado, la pertenencia del problema de asignación cuadrático y el problema de balance curricular a la clase NP-duro y el supuesto del autor de este trabajo de que el problema de balance de carga docente también pertenece a la clase NP-duro, sugiere la utilización de algoritmos heurísticos/meta-heurísticos para la solución de instancias reales del problema.

En las dos secciones siguientes se comentan brevemente el enfoque clásico para la solución de problemas cuya formulación matemática conduzca a un modelo de programación lineal y el enfoque heurístico/meta-heurístico. En particular para este último, se comentan tres algoritmos ampliamente utilizados en la solución de los problemas de confección de horarios.

1.3. Algoritmos de la programación lineal

La programación lineal (PL) se enfoca en la solución de problemas de optimización cuya modelación posee la siguiente estructura.

$$
\begin{aligned}
&\text{máx ó mín} \sum_{i=1}^{n} c_i x_i \\
&\text{sujeto a :} \\
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
&\quad\vdots \qquad\quad \vdots \qquad\qquad \vdots \qquad \vdots \\
&a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \\
&x_1,\, x_2, \cdots,\, x_n \geq 0
\end{aligned}
$$

El algoritmo Simplex, propuesto inicialmente por Dantzig (Dantzig, 1951), ha sido extensamente utilizado en la solución de instancias del problema anterior. Este algoritmo recorre los vértices del poliedro de soluciones factibles en búsqueda de la evaluación óptima. Esta solución se detecta mediante el criterio de optimalidad (teorema 2) en (Dantzig, 1951).

En términos geométricos, la región factible es un politopo convexo. Hay una caracterización simple de los puntos extremos o vértices de este politopo, a saber, x es un punto extremo si

y sólo si el subconjunto de vectores de columna correspondiente a los elementos no nulos de x son linealmente independientes. En este contexto, x se conoce como una solución básica factible (SBF).

Se puede demostrar que para un problema de programación lineal en forma estándar, si la función objetivo tiene un valor mínimo en la región factible entonces toma este valor en (al menos) uno de los puntos extremos. Esto reduce el problema a un número finito de cálculos ya que existe un número finito de puntos extremos, pero el número de puntos extremos es demasiado extenso para todos, excepto los programas lineales más pequeños.

Además se puede demostrar que si un punto extremo no es un punto mínimo de la función objetivo entonces existe una arista que contiene el punto de modo que la función objetivo es estrictamente decreciente en el borde alejándose del punto. Si el borde es finito entonces el borde se conecta a otro punto extremo en el que la función objetivo tiene un valor menor, de lo contrario la función objetivo no está acotada inferiormente y el problema no tiene solución. El algoritmo simplex aplica esta idea recorriendo los bordes del politopo hasta los puntos extremos con valores objetivos cada vez más bajos. Este procedimiento continúa hasta que se alcanza el valor mínimo o un borde no acotado es visitado, y se concluye que el problema no tiene solución. El algoritmo siempre termina porque el número de vértices en el politopo es finito; además, ya que se salta entre vértices siempre en la misma dirección (la de la función objetivo), se espera que el número de vértices visitados será pequeño.

La solución de un problema de programación lineal se lleva a cabo en dos etapas. En la primera etapa, conocida como Fase I, se encuentra un punto extremo de partida. Dependiendo de la naturaleza del problema esto puede ser trivial, pero en general se resuelve mediante la aplicación del algoritmo simplex a una versión modificada del problema original original. Los posibles resultados de la Fase I son: o bien se encuentra una solución básica factible o la región factible está vacía. En este último caso, el problema de programación lineal se llama no factible. En la segunda etapa, o Fase II, el algoritmo simplex se aplica utilizando la solución factible básica que se encontró en la Fase I como punto de partida. Los posibles resultados de la Fase II son: se obtuvo una solución factible básica óptima o un borde infinito en el que la función objetivo es no acotada inferiormente.

La transformación de un problema de programación lineal arbitrario a uno en la forma estándar puede llevarse a cabo de la siguiente manera. En primer lugar, para cada variable con una cota inferior distinta de cero, se introduce una nueva variable que representa la diferencia entre la variable y su cota inferior. La variable original puede entonces ser

eliminada por sustitución.

En segundo lugar, para cada restricción de desigualdad restante, se introduce una nueva variable, llamada variable de holgura, para cambiar la restricción a una restricción de igualdad. Esta variable representa la diferencia entre los miembros de la desigualdad y se asume que es no negativa.

En tercer lugar, cada variable sin restricción de no negatividad se elimina del problema de programación lineal. Esto puede hacerse sustituyendo la variable en cuestión con la diferencia de dos variables no negativas.

La operación geométrica de pasar de una solución básica factible a una solución básica factible adyacente se implementa como una operación de pivote. En primer lugar, se selecciona un elemento de pivote distinto de cero en una columna no básica. La fila que contiene este elemento se multiplica por su recíproco para cambiar este elemento a 1, y luego se añaden múltiplos de la fila de las otras filas para cambiar las otras entradas en la columna a 0. El resultado es que, si el elemento de pivote está en la fila i, a continuación, la columna se convierte en la columna de i-ésima de la matriz identidad. La variable para esta columna es ahora una variable básica, en sustitución de la variable que correspondía a la columna de i-ésima de la matriz identidad antes de la operación. En efecto, la variable correspondiente a la columna pivote entra en el conjunto de variables básicas y se llama la variable de entrada, y la variable siendo sustituida abandona el conjunto de variables básicas y se llama la variable de salida.

Puesto que la variable de entrada será, en general, incrementada de 0 a un número positivo, el valor de la función objetivo se reducirá si la derivada de la función objetivo con respecto a esta variable es negativa. De manera equivalente, el valor de la función objetivo se reduce si se selecciona la columna pivote de modo que la entrada correspondiente en la fila del objetivo es positiva.

Si hay más de una columna de modo que la entrada en la fila de la función objetivo es positiva, entonces la elección de cuál añadir al conjunto de variables básicas es de alguna manera arbitraria y varias reglas para seleccionar variables de entrada han sido desarrolladas.

Si todas las entradas en la fila de la función objetivo son menores que o igual a 0, entonces no hay opción para seleccionar una variable de entrada y la solución es óptima.

Una vez que la columna pivote ha sido seleccionada, la elección de la fila de pivote está determinada en gran medida por el requisito de que la solución resultante sea factible.

En primer lugar, solo las entradas positivas en la columna pivote se consideran ya que esta garantiza que el valor de la variable de entrada será no negativo. Si no hay entradas positivas en la columna pivote entonces la variable de entrada puede tomar cualquier valor no negativo con el resto de la solución factible.

A continuación, la fila de pivote debe seleccionarse de manera que todas las otras variables básicas sigan siendo positivas. Esto ocurre cuando el valor resultante de la variable de entrada está en un mínimo.

Si además se requiere que algunos de (posiblemente todos) los valores de las variables en la solución óptima sean enteros no negativos se está en presencia de modelos de la programación lineal entera mixta (PLEM) o programación lineal en enteros puros (PLE) cuando todas las variables decisión se requieren sean enteras. En estos casos pueden emplearse el algoritmo Branch and Bound (Ramificación y acotación) (Land y Doig, 1960), planos cortantes de Gomory (Gomory, 1958, 1960) o combinaciones de éstos (Padberg y Rinaldi, 1991). La idea básica que siguen estos algoritmos es la solución de un problema relajado o un conjunto de problemas relajados, es decir, eliminando la restricción de solución entera. Por ejemplo, los algoritmos de planos cortantes generan nuevas restricciones que hacen infactible la solución no entera del problema relajado pero que son cumplidas por la solución entera del problema original. Este procedimiento de corte de la región factible del problema relajado continúa hasta que se satisface la condición de entera para las variables correspondientes en la solución.

El algoritmo de Ramificación y acotación es un algoritmo general para la búsqueda de soluciones óptimas de diversos problemas de optimización, sobre todo en la optimización discreta y combinatoria. Un algoritmo de Ramificación y acotación consiste en una enumeración sistemática de todas las soluciones de candidatas. Una gran cantidad de estas soluciones candidatas se descartan en masa mediante el uso de cotas superiores e inferiores estimadas para la función objetivo.

El método fue propuesto por primera vez por AH Tierra y AG Doig (Land y Doig, 1960) en 1960 para la programación discreta.

Si se supone que el objetivo es encontrar el valor mínimo de una función $f(x)$. Donde x toma valores sobre un conjunto S de soluciones admisibles o soluciones candidatas (el espacio de búsqueda o región la factible).

Un procedimiento de Ramificación y acotación requiere dos procedimientos. El primero es un procedimiento de división que, dado un conjunto S de candidatos, devuelve dos o más

pequeños conjuntos S_1, S_2, cuya unión cubre a S. Nótese que el mínimo de $f(x)$ sobre S es mín$\{v_1, v_2, \ldots\}$, donde cada v_i es el mínimo de $f(x)$ sobre S_i . Este paso se conoce como *ramificación*, puesto que de su aplicación recursiva se obtiene una estructura de árbol (el árbol de búsqueda) cuyos nodos son los subconjuntos de S.

La segunda herramienta es un procedimiento que calcula las cotas superior e inferior para el valor mínimo de $f(x)$ sobre un subconjunto dado de S . Este paso se conoce por *acotación*.

La idea clave del algoritmo de Ramificación y acotación es: si la cota inferior por algún nodo del árbol (conjunto de candidatos) N_1 es mayor que la cota superior para algún otro nodo N_2, entonces el nodo N_1 puede ser descartado de la búsqueda de forma segura. Este paso se llama *poda*, y normalmente se implementa mediante el mantenimiento de una variable global l (compartida entre todos los nodos del árbol) que registra la mínima cota superior vista entre todas las subregiones analizadas hasta el momento. Cualquier nodo cuya cota inferior es mayor que l puede ser descartado.

La recursión se detiene cuando el conjunto de soluciones candidatas S se reduce a un solo elemento, o cuando la cota superior para el conjunto S coincide con su cota inferior. De cualquier manera, cualquier elemento de S será un mínimo de la función $f(x)$ dentro de S.

El algoritmo de Ramificación y acotación también pueden ser utilizado como base para diversas heurísticas . Por ejemplo, se puede desear detener la ramificación cuando la brecha entre las cotas superior e inferior se haga menor que un cierto umbral. Esto se utiliza cuando la solución es lo suficientemente "buena" para los propósitos prácticos y pueden reducirse en gran medida los cálculos requeridos. Este tipo de solución es especialmente aplicable cuando la función objetivo utilizada contiene ruido o es el resultado de estimaciones estadísticas y por tanto no se conoce con precisión, sino que es conocida solo dentro de un rango de valores con una determinada probabilidad.

Las formulaciones 2.7 y 2.16 del problema de balance de carga docente se ajustan al modelo PLE por lo tanto pueden, teóricamente, aplicarse los algoritmos diseñados al efecto. No obstante, la pertenencia del modelo 2.16 a la clase NP-duro no permite su solución utilizando estos algoritmos para instancias reales del problema. Esto justifica la utilización de los algoritmos que se describen en la siguiente sección.

1.4. Algoritmos heurísticos/meta-heurísticos para la optimización

En esta sección se revisan algunos de los algoritmos heurísticos/meta-heurísticos basados en búsqueda local y en inteligencia de enjambre. Antes de analizar estos algoritmos conviene aclarar qué se entiende por algoritmo heurístico/meta-heurístico, búsqueda local e inteligencia de enjambre.

Las heurísticas son algoritmos que localizan buenas soluciones sin tomar en cuenta si la solución es correcta u óptima (Michalewicz y Fogel, 2000). Las meta-heurísticas son algoritmos de solución que orquestan una interacción entre los procedimientos locales de mejora y estrategias de nivel superior para crear un proceso capaz de escapar de óptimos locales y la realización de una búsqueda robusta en un espacio de soluciones (Glover y Kochenberger, 2003).

De acuerdo con (Panigrahi, Shi, y Lim, 2011), la inteligencia de enjambre o colectiva es una colección de algoritmos bio-inspirados agrupados en lo que se conoce como computación evolutiva. Son algoritmos basados en poblaciones, en los cuales los individuos de una población (soluciones candidatas potenciales) cooperan entre ellos para, estadísticamente, mejorar durante generaciones y eventualmente encontrar buenas soluciones. En esta colección se encuentran los algoritmos genéticos (GAs)[8] (Holland, 1975), optimización mediante enjambre de partículas (PSO) (Eberhart y Kennedy, 1995), entre otros.

Los algoritmos de búsqueda local comienzan a partir de una de solución inicial e iterativamente tratan de reemplazar la solución actual por una mejor en una vecindad definida apropiadamente a partir de la solución actual (Blum y Roli, 2003). Dentro de los métodos de búsqueda local se tiene a búsqueda tabú (TS) que fue inicialmente concebido para resolver problemas combinatorios (Glover y McMillant, 1986).

[8]El autor de este trabajo conservará el uso de siglas en inglés para hacer referencia a los algoritmos heurísticos/meta-heurísticos a través de todo el documento.

1.4.1. Optimización mediante Búsqueda Tabú

TS fue desarrollado por Glover en 1986. A diferencia de los algoritmos anteriores en los que se mantiene una población de soluciones candidatas, TS es un algoritmo de búsqueda local en el cual a partir de una solución inicial se va, iterativamente, mejorando la solución mediante la exploración de una vecindad de la solución actual. Un elemento fundamental en TS son los movimientos tabú. TS mantiene una lista dinámica (Lista Tabú) con los movimientos tabú, el propósito de esta lista es evitar que el algoritmo caiga en ciclos puesto que cada nuevo movimiento si está en la lista es descartado, al menos durante varios ciclos de ejecución (esto es conocido como permanencia tabú del movimiento). Los movimientos tabú más usuales incluyen almacenar las últimas transformaciones de la solución actual y prohibir las transformaciones inversas. La lista tiene usualmente una longitud fija y es actualizada en cada ciclo insertando los movimientos tabú y eliminando los más antiguos si fuese necesario. Véase (Gendreau, 2003) para una introducción más completa.
La estructura general y por la que el autor de este trabajo desarrolla el algoritmo propuesto, es la siguiente tomada de (Gendreau, 2003):
Notación:

- X: Solución actual.
- X^*: Mejor solución conocida.
- f^*: Valor de X^*.
- $N(X)$: Vecindad de X.
- $\widetilde{N}(X)$: Subconjunto admisible de de $N(X)$.
- T: Lista tabú.

Algoritmo A: Esquema básico de la Búsqueda Tabú.

$X := X_0$;
$X^* := X_0$;
$f^* := f(X^*)$;
$T := \emptyset$;
while *no se satisfaga la condición de parada* **do**
    Seleccionar X en $argmin[f(X')]$ $X' \in \widetilde{N}(X)$;
    if $f(X) < f^*$ **then**
        $f^* := f(X)$;
        $X^* := X$;
    end
    Guardar el movimiento tabú correspondiente en T;
    Eliminar el movimiento tabú más antiguo si es necesario;
end

TS es quizás junto a GAs uno de los algoritmos que mayor impacto ha tenido en la solución de problemas de optimización combinatoria. Una aplicación al problema clásico de ruteo de vehículos puede consultarse en (Gendreau, 2003). Algunas aplicaciones de TS a la generación de horarios pueden ser revisadas en (Burke, Kendall, y Soubeiga, 2003), (Cartaya, 2009) y en (Naseem y Shengxiang, 2010) y (Cartaya, 2013), donde se combina con GAs. La combinación de GAs y TS propuesta por (Cartaya, 2013) resulta esencialmente la propuesta por los autores en (Naseem y Shengxiang, 2010), no obstante resulta meritorio el hecho de haberlo aplicado en un problema real, la confección de horarios en la Universidad de Cienfuegos, además de integrarlo en una aplicación WEB actualmente en funcionamiento que permite la consulta de los horarios de estudiantes, profesores y aulas. Sin dudas el trabajo de (Cartaya, 2013) constituye un avance importante para la posterior integración con el sistema SIGENU. El algoritmo propuesto combina efectivamente la buena exploración del espacio de búsqueda brindada por GAs con la explotación local de TS. Esta unión permite obtener un algoritmo que, de acuerdo con los resultados de las pruebas numéricas obtenidas por los autores en (Naseem y Shengxiang, 2010), resulta competitivo con otras variantes conocidas sobre los problemas de horarios planteados en la literatura para la comparación de estos tipos de algoritmos.

En (Taillard, 1991) los autores proponen una adaptación de TS para el problema de asignación cuadrática de la sección 1.2.2. Esta adaptación es eficiente y robusta, requiere menos complejidad y menor número de parámetros que las adaptaciones propuestas por

otros autores anteriores. Con el fin de mejorar la velocidad de convergencia del algoritmo TS, los autores proponen dos métodos de paralelización y se muestra la eficiencia alcanzada con cada uno de estos métodos utilizando un número de procesadores proporcional a la magnitud del problema a resolver.

Los autores mejoran las soluciones publicadas a muchos de los problemas más grandes que han sido planteados en la literatura. Además confirman las soluciones encontradas con anterioridad a problemas más pequeños, lo cual hace sospechar que se tratan efectivamente de las soluciones óptimas de esos problemas. Además, se propone una manera fácil de generar problemas aleatorios y buenas soluciones a estos problemas, cuyos tamaños oscilan entre 5 y 100.

1.4.2. Los algoritmos poblacionales: Algoritmos genéticos y Enjambre de partículas

Los GAs son algoritmos de búsqueda basados en los principios de la selección natural y la genética. En los GAs se codifican las variables de decisión de un problema de búsqueda en cadenas de longitud finita. Las cadenas son soluciones candidatas para el problema y se conocen como cromosomas, las posiciones se conocen como los genes y los valores de los genes se denominan alelos. Para desarrollar buenas soluciones y aplicar la selección natural, se utiliza una medida que distingue las buenas soluciones de malas soluciones. La medida podría ser una función objetivo o una simulación por computadora. Básicamente, la medida de adaptación debe determinar la aptitud relativa de una solución candidata, que posteriormente será utilizada por el algoritmo para evolucionar las buenas soluciones. Otro concepto es la noción de población. A diferencia de los algoritmos clásicos de búsqueda, los algoritmos genéticos se basan en una población de soluciones candidatas. El tamaño de esta población determina en gran parte la escalabilidad y el rendimiento de los algoritmos genéticos (Burke y Kendall, 2005).
Los pasos básicos del algoritmo son los siguientes (Burke y Kendall, 2005):

1. **Inicialización**. Se genera aleatoriamente una población inicial de soluciones candidatas en el espacio de búsqueda.

2. **Evaluación**. Una vez que la población es inicializada o se crea una población de descendientes, todas las soluciones candidatas son evaluadas.

3. **Selección**. La idea fundamental de la selección consiste en escoger las mejores soluciones teniendo en cuenta la medida de adaptación. El proceso de selección se efectúa al azar mediante un procedimiento que favorezca a las mejores soluciones. Muchos procedimientos de selección han sido propuestos, incluyendo el basado en la ruleta sesgada en el cual la probabilidad de que una solución sea seleccionada es proporcional a su medida de adaptación.

4. **Recombinación**. La recombinación combina partes de dos o más soluciones padres para crear nuevas y posiblemente mejores soluciones (descendientes). Este procedimiento es llevado a cabo a través de *operadores de cruzamiento*.

5. **Mutación**. La mutación modifica aleatoriamente una solución. Hay muchas maneras, pero generalmente involucran cambios en uno o más rasgos de los individuos. Este procedimiento es llevado a cabo a través de *operadores de mutación*.

6. **Reemplazo**. La población descendiente creada por selección, recombinación y mutación sustituye a la población original de padres. Existe técnicas de reemplazo tales como reemplazo simple, reemplazo elitista y otras que se colocan entre los anteriores.

7. Se repiten los pasos del 1 al 6 hasta que una condición de terminación sea satisfecha.

Existen varios tipos de operadores de cruzamiento. Entre ellos, los más utilizados son: Cruzamiento por un punto, Cruzamiento por dos puntos, Cruzamiento por corte y emplame y Cruzamiento uniforme.

Cruzamiento por un punto

Se selecciona un único punto de cruce en las cadenas de las soluciones padres. Todos los datos más allá de ese punto, ya sea en cadena organismo se intercambian entre los dos soluciones padres. Las soluciones resultantes son los hijos que se muestran en la figura 1.1:

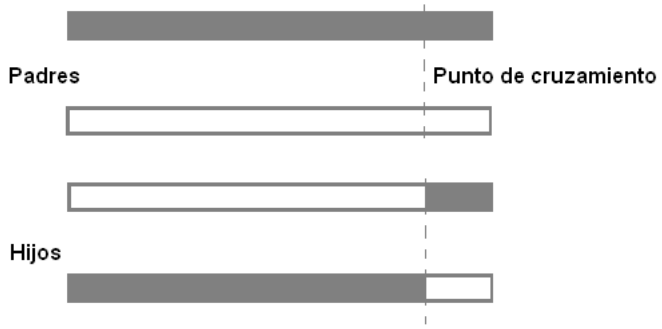


Figura 1.1: Cruzamiento por un punto

Cruzamiento por dos puntos

El cruzamiento por dos puntos requiere dos puntos para ser seleccionados en las cadenas de los padres. La información entre estos dos puntos se intercambia entre las soluciones padres obteniéndose dos soluciones hijos que se muestran en la figura 1.2:

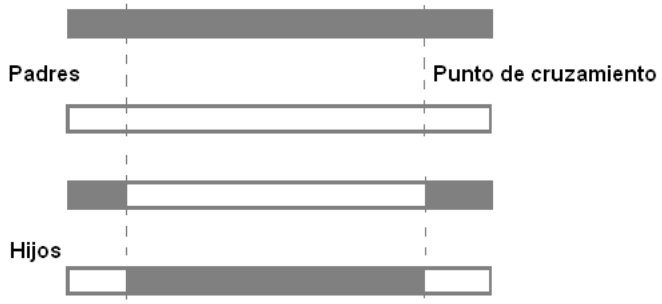


Figura 1.2: Cruzamiento por dos puntos

Cruzamiento por corte y emplame

Otra variante de cruzamiento emplea el enfoque de corte y empalme. Esta variante da como resultado un cambio en la longitud de las cadenas de las soluciones hijos. La razón de esta diferencia es que cada cadena de las soluciones padres tiene una opción separada para el punto de cruzamiento tal y como se muestra en la figura 1.3:

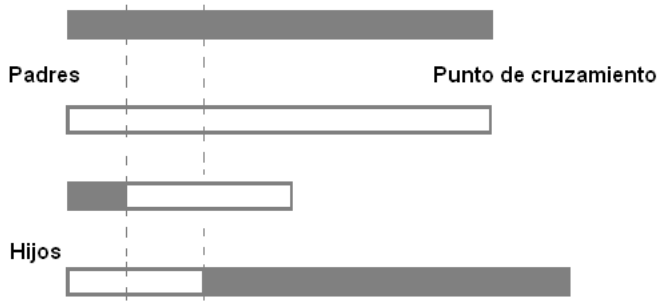


Figura 1.3: Cruzamiento por corte y empalme

Cruzamiento uniforme

El cruzamiento uniforme utiliza una relación de intercambio fija entre los dos padres. A diferencia del cruzamiento por un y dos puntos, el cruzamiento uniforme permite a los cromosomas de los padres contribuir al nivel de genes en lugar de al nivel de segmentos.

Si la proporción de intercambio es de 0.5, la descendencia tiene aproximadamente la mitad de los genes de primer padre y la otra mitad del segundo padre, aunque los puntos de cruzamiento pueden ser seleccionados en forma aleatoria, como se ve muestra en la figura 1.4:

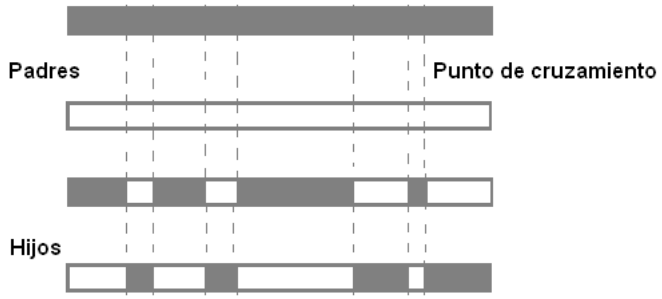


Figura 1.4: Cruzamiento uniforme

Operadores de mutación

La mutación es un operador genético utilizado para mantener la diversidad genética de una población de cromosomas de una generación a la siguiente. Es análogo a una mutación biológica. La mutación altera los valores de uno o más genes en un cromosoma a partir de su estado inicial. Al aplicar el operador de mutación, la solución puede cambiar por completo con respecto a la solución anterior. De ahí que los GAs pueden llegar a una mejor solución mediante la mutación. La mutación se produce durante la evolución de la población de

acuerdo una probabilidad de mutación definida por el investigador. Esta probabilidad se debe establecer baja. Si es demasiado alta, la búsqueda se convertirá básicamente en una búsqueda aleatoria. Los operadores más utilizados en la práctica son: La mutación de los bits[9] y el cambio del valor binario.

Mutación de bits

La mutación de los bits en la solución consiste en el cambio del valor binario de los genes de acuerdo con la probabilidad de mutación establecida por el investigador. El resultado de esta operación se muestra en la tabla 1.1.

Tabla 1.1: Resultado del operador de mutación: Mutación de bits

1	0	0	0	0
		↓	↓	
1	0	1	0	1

Cambio del valor binario

Este operador elige aleatoriamente uno de los genes del cromosoma y cambia su valor binario. Por ejemplo, si en el primer cromosoma de la tabla 1.1 hubiese sido seleccionado el cuarto gen, entonces el resultado del operador sería la solución mostrada en la tabla 1.2.

Tabla 1.2: Resultado del operador de mutación: Cambio del valor binario

1	0	0	0	0
			↓	
1	0	0	1	0

Los GAs han encontrado aplicación en la solución de diversos problemas de la ciencia y la ingeniería. Véase (Al-milli, 2010) y (Cartaya, 2013), aplicaciones al problema de la generación de horarios y (Gallardo y Lowther, 2000), aplicado a la optimización de dispositivos electromagnéticos. También se han aplicado en la optimización de los procesos de fermentación de Saccharomyces cerevisiae, un tipo de levadura (Jones, 2006).

[9]Bit es el acrónimo de dígito binario. Un bit es un dígito en el sistema de numeración binario.

Optimización mediante enjambre de partículas

PSO es similar a un algoritmo genético debido a que el sistema es inicializado con una población de partículas (soluciones) aleatorias (Eberhart y Kennedy, 1995). Sin embargo se diferencian en que además a cada partícula se le asigna una velocidad aleatorizada. Cada partícula memoriza la mejor posición que ha visitado en el espacio de búsqueda (personal best) la cual es denotada por *pbest*. La mejor solución alcanzada por el enjambre (global best) es también memorizada y se denota por *gbest*. La optimización se lleva a cabo cambiando la velocidad y acelerándo las partículas hacia su *pbest* y la mejor solución alcanzada por el enjambre *gbest*. Este procedimiento se realiza mediante las ecuaciones:

$$v_i^{(t+1)} := w \cdot v_i^{(t)} + c_1 \cdot unif(0,1) \cdot \left[pbest_i - x_i^{(t)}\right] + c_2 \cdot unif(0,1) \cdot \left[gbest - x_i^{(t)}\right]$$
$$x_i^{(t+1)} := x_i^{(t)} + v_i^{(t+1)}$$

El parámetro w se conoce como *peso de inercia*, c_1 es el peso asociado a la memoria e indica cuánto se deja influenciar la partícula por su mejor posición, c_2 favorece el comportamiento social del enjambre ya que indica cuánto se deja influenciar la partícula por la mejor posición alcanzada por el enjambre. Es usual tomar $w = 0{,}2$ y $c_1 = c_2 = 2$. La bibliografía reporta que deben tomarse c_1 y c_2 tal que $c_1 + c_2 \leq 4$, aunque en realidad los valores de estos parámetros dependen del problema en cuestión.

PSO también ha sido utilizado en innumerables aplicaciones de la ciencia y la ingeniería. Véase (Deris, Zaiton, y Hashim, 2009), (Alinia, Taghi, y Baghmisheh, 2012) aplicado al problema de confección de horarios, (Yin, 2004), una versión de PSO para optimización discreta aplicada a la aproximación poligonal de curvas digitales y (Leong y Yen, 2008), una aplicación a la optimización multiobjetivo.

En (Paquet y Engelbrecht, 2003) los autores proponen una estrategia basada en PSO para la optimización de funciones sobre un conjunto de restricciones lineales. Básicamente se propone inicializar un conjunto de soluciones candidatas factibles y luego utilizar PSO con esta población factible. La propia naturaleza vectorial de las ecuaciones de PSO garantiza la factibilidad durante el proceso de optimización. Una variante de este enfoque propuesta por el autor de este trabajo y que no ha sido publicada resulta la siguiente:

Dado el siguiente problema:

$$\begin{aligned} &\text{mín } f(x) \quad x \in \mathbb{R}^n \\ &\text{sujeto a: } Ax = b \quad A \in \mathbb{R}^{m \times n} \text{ y } b \in \mathbb{R}^m \end{aligned}$$

La propuesta del autor de este trabajo se basa en la preservación de la factibilidad empleando una base del espacio nulo de la matriz A. Bajo este enfoque, cualquier solución del sistema $Ax = b$ puede escribirse en la forma $x = p + \sum_{i=1}^{n} \lambda_i \cdot s_i$, donde $\{s_i\}$ constituye una base del espacio nulo de A y p es cualquier solución del sistema $Ax = b$. De esta forma se va de un problema restringido linealmente en la variable de decisión x a otro problema irrestricto en términos de los coeficientes de la combinación lineal $\sum_{i=1}^{n} \lambda_i \cdot s_i$. El autor emplea las mismas ecuaciones originales de PSO:

$$v_i^{(t+1)} := w \cdot v_i^{(t)} + c_1 \cdot unif(0,1) \cdot \left[pbest_i - \lambda_i^{(t)}\right] + c_2 \cdot unif(0,1) \cdot \left[gbest - \lambda_i^{(t)}\right]$$
$$\lambda_i^{(t+1)} := \lambda_i^{(t)} + v_i^{(t+1)}$$

cambiando solamente x por λ. Los resultados numéricos muestran que el método propuesto es efectivo en las funciones de prueba analizadas: la función de Rosenbrock (Rosenbrock, 1960) y la función de Himmelblau (Andrei, 2008), dos funciones propuestas para la prueba de algoritmos de optimización.

1.5. Complejidad algorítmica y estrategias para la solución de problemas NP-duros

En secciones anteriores se han utilizado los términos NP y NP-duro sin definir apropiadamente sus significados. En esta sección se aborda brevemente parte de la teoría de la complejidad con el propósito de definir ambos términos.

De acuerdo con (Bai, 2005), la complejidad de un algoritmo se mide en términos de la complejidad de tiempo y la complejidad de espacio. La complejidad de tiempo es una medida del tiempo necesario para ejecutar el algoritmo, dado un tamaño determinado de la entrada. Esta medida está dada por un conjunto de funciones elementales que incluyen las funciones: constante, logarítmica, polinómica y la exponencial. La complejidad de espacio es una medida cuánto espacio de almacenamiento requiere el algoritmo.

La complejidad de un problema está dada por la complejidad de tiempo del algoritmo más eficiente que lo soluciona (Michael y Johnson, 1979). Un problema es "manejable" si existe un algoritmo que lo soluciona en tiempo polinomial, de otro modo el problema es "no manejable". En este caso puede suceder que el problema no tenga solución o que se necesita un tiempo exponencial para solucionarlo.

La clase P agrupa los problemas que se pueden resolver por un algoritmo determinístico en

tiempo polinomial y la clase NP agrupa los problemas que pueden ser resueltos en tiempo polinomial por un algoritmo no determinístico. Se sabe que P $\subseteq$ NP pero, aunque no se ha demostrado, se cree que P$\neq$NP. Como consecuencia de esto existe un grupo de problemas que no pertenecen a la clase P (son problemas "no manejables"). Los problemas más difíciles de la clase NP son los llamados problemas NP-completos, estos problemas fueron identificados inicialmente por Stephen A. Cook en (Cook, 1971). Otra clasificación resulta NP-duro, estos problemas se consideran al menos tan difíciles como los NP-completos, aunque no se ha probado que son problemas "no manejables".

En el caso del problema de balance de carga docente, se supone que se trata de un problema "no manejable" ya que los experimentos numéricos del capítulo tres así lo sugieren.

Cuando se está enfrentando estos tipos de problemas los algoritmos más eficientes para la solución de instancias grandes resultan los discutidos en la sección 1.4. No obstante, las investigaciones continúan para dotar a los métodos exactos de vías eficientes para explorar el espacio de búsqueda y acelerar la convergencia hacia posibles óptimos globales. Con frecuencia los algoritmos exactos se utilizan en combinación con los algoritmos heurísticos/meta-heurísticos para de esta forma aprovechar las ventajas que ambos enfoques presentan. En este sentido, se instrumentan heurísticas que permiten, a los algoritmos exactos, descartar regiones donde con seguridad no se encuentra el óptimo global. Así por ejemplo, en (Nagar, Heragu, y Haddock, 1996) se combinan el algoritmo de Ramificación y acotación con GAs para resolver un problema de programación de la línea de flujo de dos máquinas. La meta-heurística utiliza un procedimiento de Ramificación y acotación para generar un poco de información, que a su vez se utiliza para guiar la búsqueda de soluciones cercanas a la óptima de un algoritmo genético. Los criterios considerados son el tiempo entre el inicio y el fin de las secuencias de trabajo y el tiempo promedio del flujo de trabajo. El problema tiene aplicaciones en entornos de líneas de flujo donde la administración está interesada en la reducción del período de tiempo para completar el ciclo del proceso y los tiempos de inactividad de forma simultánea. Los autores, además, comparan tres versiones del algoritmo: Ramificación y acotación puro, algoritmo genético puro, y la heurística propuesta. Los resultados indican que el enfoque combinado y sus versiones modificadas son mejores que cualquiera de las estrategias puras.

Las meta-heurísticas también son combinadas entre ellas, como ya ha sido visto, para aprovechar las ventajas que cada una ofrece en la solución de un problema determinado. Por ejemplo, en (Xia y Wu, 2006) los autores proponen un nuevo algoritmo de aproximación para el problema de encontrar el tiempo entre el inicio y el fin de las

secuencias de trabajo mínimo en el entorno de la programación de líneas de flujo de máquinas. El nuevo algoritmo se basa en el principio de la optimización seguido por el algoritmo PSO. Según los autores, PSO combina búsquedas locales (por experiencia propia) y la búsqueda global (por la experiencia de la mejor partícula del enjambre), y posee una alta eficiencia de búsqueda. Los autores combinan a PSO con el algoritmo de recocido simulado (SA), el cual emplea cierta probabilidad para evitar quedar atrapado en un óptimo local. Al combinar razonablemente estos dos algoritmos de búsqueda diferentes, los autores desarrollaron un algoritmo general de optimización híbrida rápido y fácil de implementar denominado HPSO. La eficacia y la eficiencia del algoritmo basado en PSO propuesto se demuestran mediante su aplicación a algunos problemas de programación de líneas de flujo de máquinas. La comparación con otros resultados en la literatura indica que el algoritmo basado en PSO resulta una variante viable y eficaz para el problema de programación de líneas de flujo de máquinas.

1.6. Conclusiones del capítulo

- Para confeccionar los horarios docentes en las universidades cubanas primero se confeccionan los balances de carga docente. Estos últimos pueden ser considerados problemas tan difíciles de resolver como los primeros.
- Los algoritmos de asignación clásicos revisados no presentan similitudes destacables con el problema de balance de carga docente.
- Los algoritmos de la programación lineal en enteros puros no resuelven instancias reales del problema de balance de carga docente en un tiempo computacionalmente aceptable.
- Los algoritmos heurísticos/meta-heurísticos constituyen una alternativa viable para la solución aproximada del problema de horarios docentes y el problema de balance de carga docente.

Capítulo 2

Modelación matemática del problema de balance de carga docente

Introducción

En el capítulo anterior fue descrito el proceso de planificación docente en la Universidad de Cienfuegos. En particular se hizo énfasis en el problema de balance de carga docente y su vínculo con el problema de horarios docentes. El propósito de este capítulo es la formulación de un modelo general para el problema de balance de carga docente en la Universidad de Cienfuegos y la selección de un algoritmo de solución apropiado. En este sentido se realizan simplificaciones del problema que pueden estar completamente justificadas en algunos casos. Estas simplificaciones permiten ir de lo particular a lo general durante la etapa de modelación. El modelo matemático de la sección 2.1 constituye una primera aproximación que simplifica el problema asumiendo turnos de clase de duración constante. El modelo matemático de la sección 2.2 resulta un modelo general que no puede ser resuelto, para instancias reales del problema, por los métodos clásicos revisados en el capítulo uno. Es por esta razón que en la sección 2.3 se plantea un modelo matemático heurístico general para el problema, tomando elementos convenientes de las formulaciones anteriores. La solución del modelo heurístico general con el algoritmo TS y los resultados numéricos se exponen en el capítulo tres.

2.1. Modelo en enteros puros para turnos de duración constante

La formulación de un modelo matemático para el problema de balance de carga docente en la Universidad de Cienfuegos tiene que satisfacer los requisitos de los planificadores docentes identificados anteriormente. La combinación de estos requisitos permite manejar un grupo de situaciones que se presentan habitualmente en la planificación docente y que afectan el desarrollo del proceso docente educativo.

Por ejemplo, en la etapa definida para la revisión de los exámenes de ingreso a la educación superior los profesores designados para los tribunales de Matemática, Español e Historia no pueden impartir docencia en esa etapa pues están completamente enfrascados en la revisión de estos exámenes. Si esta etapa está definida, por ejemplo de la siguiente manera: Matemática semana s, Español semana $s + 1$ e Historia semana $s + 2$, entonces en los balances de carga docente que involucren asignaturas impartidas por estos profesores es necesario bloquear estas semanas y no asignar turnos de clase a estas asignaturas. Es habitual también que el fondo de tiempo semanal varíe en las distintas semanas del período de planificación. Estas variaciones del fondo de tiempo semanal están dadas por la planificación de actos políticos, reuniones, días feriados, entre otras situaciones que de manera directa o indirecta, y ya sean planificadas o no, afectan la docencia.

Una situación similar se tiene cuando ocurren desastres naturales. En estos casos la planificación docente debe reestructurarse para poder asumir, en la etapa de recuperación, toda la carga docente que fue afectada durante la situación de desastre. En el contexto de la planificación docente, realmente no se trata de una transición trivial, pues como ha sido explicado, la confección de todos los balances de carga docente toma aproximadamente tres meses de trabajo y además luego deben prepararse los nuevos horarios de clase. Es decir, la modelación matemática tiene que considerar que debe darse respuesta rápida al decisor.

Los planificadores organizan el balance de carga docente de un grupo en una tabla donde las filas coinciden con las semanas y las columnas con las asignaturas del semestre. Inicialmente se asume que todos los turnos de clase tienen una duración constante (generalmente dos horas).

Se definen:

Un conjunto de asignaturas $I = \{1, 2, \cdots, n\}$ y de semanas $J = \{1, 2, \cdots, m\}$.

Variables del modelo

X_{ij}: Número de encuentros a impartir de la asignatura i en la semana j.

d_j: Desviaciones por debajo del balance en la semana j.
D_j: Desviaciones por encima del balance en la semana j.

Parámetros del modelo

Sean h, p_i y Q_j, la duración de los turnos de clase (generalmente dos horas), las cantidad de turnos de clase planificados en el P1 de la asignatura i y las horas clases disponibles en la semana j respectivamente.

Restricciones del modelo

La cantidad de turnos de clase a impartir de la asignatura i durante la semana j tiene que estar entre la cantidad mínima y la máxima de turnos de clase permitidos.

$$lb_{ij} \leq X_{ij} \leq ub_{ij} \quad \forall i \in I \quad \text{y} \quad \forall j \in J \tag{2.1}$$

Las variables enteras asociadas al balance son no negativas.

$$d_j,\ D_j \geq 0 \quad \forall j \in J \tag{2.2}$$

La siguiente restricción garantiza que se respete la cantidad de turnos de clase planificados en el P1 de cada asignatura.

$$\sum_{j \in J} X_{ij} = p_i \quad \forall i \in I \tag{2.3}$$

Aún cuando se persigue un balance en la carga docente es posible que en algunas semanas del semestre las horas clases disponibles se vean afectadas. Esta situación se maneja mediante la restricción:

$$h \cdot \sum_{i \in I} X_{ij} \leq Q_j \quad \forall j \in J \tag{2.4}$$

La planificación docente en la Universidad de Cienfuegos toma poco en cuenta las consideraciones de los profesores, pero aún así, aunque no de manera general, es posible que los profesores negocien con los planificadores docentes la cantidad de turnos de clase a impartir de sus asignaturas en una o varias semanas. Esto es, obviamente, solo en casos excepcionales, pues considerarlo en demasiadas semanas afectaría el balance de carga docente. Haciendo uso de las restricciones de caja formuladas en 2.1 puede establecerse una cantidad fija de turnos de clase a impartir de una asignatura i en una semana j.

Función objetivo cuadrática

El objetivo es balancear la carga docente semanal durante todo el semestre. Esto se logra minimizando las desviaciones cuadradas de las cargas semanales respecto al promedio.

$$\text{mín} \sum_{j \in J} \left(\sum_{i \in I} X_{ij} - \frac{1}{m} \sum_{i \in I} \sum_{j \in J} X_{ij} \right)^2 \tag{2.5}$$

Para obtener un modelo de PLE equivalente se plantea:

$$m \cdot \sum_{i \in I} X_{ij} - \sum_{i \in I} \sum_{j \in J} X_{ij} + d_j - D_j = 0 \quad \forall j \in J \tag{2.6}$$

Función objetivo lineal

Minimizando la suma de las desviaciones se obtiene un resultado equivalente a 2.5:

$$\text{mín} \sum_{j \in J} (d_j + D_j) \tag{2.7}$$

Las ecuaciones 2.1, 2.2, 2.3, 2.4, 2.6 y 2.7 conforman un modelo de PLE para el problema de balance de carga docente, considerando todos los turnos de clase de duración constante.

Alcance de la aplicación del modelo

Los balances de carga docentes con turnos de clase de duración constante son bastante frecuentes en la práctica. En estos casos puede utilizarse el modelo planteado en esta sección junto a un algoritmo clásico de solución como los descritos en la sección 1.3. Debe señalarse que aún en este caso se trata de un problema combinatorio por lo que no hay garantía absoluta de que el algoritmo clásico resuelva todas las instancias del modelo en un tiempo computacional aceptable. El autor de este trabajo resuelve algunas instancias de este modelo de manera exacta, utilizando el software coinMP, para la validación del algoritmo propuesto; estos resultados son expuestos en el capítulo tres.

2.2. Modelo en enteros puros para turnos de duración arbitraria

Como se ha planteado con anterioridad, el modelo anterior es aplicable solo si se consideran todos los turnos de clase de duración constante. Un modelo más general, que incluye turnos de clase de distinta duración, pero que introduce dificultades para su solución es el siguiente:

Sea h_{ik} la duración del turno de clase k de la asignatura i. Los valores que puede tomar h_{ik} en la práctica son los números naturales menores o iguales que seis.

Se define:

Variables del modelo

$$X_{ijk} = \begin{cases} 1\,, & \text{el encuentro } k \text{ de la asignatura } i \text{ se imparte en la semana } j \\ 0\,, & \text{en otro caso} \end{cases} \tag{2.8}$$

Como en el caso anterior se definen:

d_j: Variable entera que representa desviaciones por debajo del balance.
D_j: Variable entera que representa desviaciones por encima del balance.

Parámetros del modelo

Sean h_{ik}, p_i y Q_j, la duración del turno de clase k de la asignatura i, las cantidad de turnos de clase planificados en el P1 de la asignatura i y las horas clases disponibles en la semana j respectivamente.

Restricciones del modelo

Las variables enteras asociadas al balance son no negativas.

$$d_j,\ D_j \geq 0 \tag{2.9}$$

La restricción siguiente garantiza que el encuentro k de cada asignatura se imparta en una semana.

$$\sum_{j \in J} X_{ijk} = 1 \quad \forall i \in I \quad \forall k \in \{1, 2, \cdots, p_i\} \tag{2.10}$$

Se tiene que garantizar que el turno de clase k de la asignatura i se imparta después del turno de clase $k-1$ o en la misma semana, para esto se plantea:

$$\sum_{j \in \{e+1, e+2, \cdots, m\}} X_{ijk-1} + X_{iek} \leq 1 \quad \forall i \in I \quad \forall k \in \{2, 3, \cdots, p_i\} \quad \forall e \in \{1, 2, \cdots, m-1\} \tag{2.11}$$

La restricción de orden 2.11 complica enormemente el modelo debido a que fuerza a la soluciones factibles a poseer una estructura matricial compleja. Este tipo de restricciones es usualmente relajada y añadida a la función objetivo multiplicada por un peso favoreciendo a aquellas soluciones donde se cumpla la restricción, pero este tipo de estrategia por supuesto no garantiza que la solución obtenida sea de hecho solución factible para el problema inicial. Como resultado de esta relajación se obtendrían soluciones en las que no se respetarían el orden de los turnos de clase de algunas (o posiblemente todas) las asignaturas, situación claramente inaceptable.

Tal y como fue visto en 2.4, pero ahora en un caso más general considerando turnos de clase de duración arbitraria, se plantea la restricción que garantiza no se exceda la cantidad de horas disponibles para una semana.

$$\sum_{i \in I} \sum_{k \in \{1, 2, \cdots, p_i\}} h_{ik} \cdot X_{ijk} \leq Q_j \quad \forall j \in J \tag{2.12}$$

De manera equivalente a 2.1 se plantea:

$$lb_{ij} \leq \sum_{k \in \{1,2,\cdots,p_i\}} X_{ijk} \leq ub_{ij} \quad \forall i \in I \quad \forall j \in J \tag{2.13}$$

De manera similar al modelo anterior el objetivo es minimizar las desviaciones cuadradas de las cargas semanales respecto al promedio:

Función objetivo cuadrática

$$\text{mín} \sum_{j \in J} \left(\sum_{i \in I} \sum_{k \in \{1,2,\cdots,p_i\}} h_{ik} \cdot X_{ijk} - \frac{1}{m} \sum_{j \in J} \sum_{i \in I} \sum_{k \in \{1,2,\cdots,p_i\}} h_{ik} \cdot X_{ijk} \right)^2 \tag{2.14}$$

Para obtener finalmente un modelo de PLE se plantea la restricción:

$$m \cdot \sum_{i \in I} \sum_{k \in \{1,2,\cdots,p_i\}} h_{ik} \cdot X_{ijk} - \sum_{j \in J} \sum_{i \in I} \sum_{k \in \{1,2,\cdots,p_i\}} h_{ik} \cdot X_{ijk} + d_j - D_j = 0 \quad \forall j \in J \tag{2.15}$$

Función objetivo lineal

Minimizando la suma de las desviaciones se obtiene el balance óptimo:

$$\text{mín} \sum_{j \in J} (d_j + D_j) \tag{2.16}$$

Las ecuaciones 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.15, 2.16 conforman un modelo de PLE para el problema de balance de carga docente considerando turnos de clase de duración arbitraria.

Alcance de la aplicación del modelo

Al inicio del capítulo se plantea que este modelo presenta nuevas dificultades para su solución. En el capítulo uno se afirma que este modelo pertenece a la clase NP-duro, de manera que no es posible resolverlo utilizando algoritmos clásicos en un tiempo computacional razonable.
Para tener una idea de la dimensión que puede alcanzar este modelo para una instancia real del problema de balance de carga docente, considérense 8 asignaturas en un semestre de 16 semanas y como promedio 30 encuentros por asignatura, lo cual es bastante conservador. Para este caso, el modelo tiene 3856 variables y 3872 restricciones.

Los modelos anteriores resueltos con algoritmos clásicos proporcionan soluciones óptimas. Sin embargo, el primer modelo no es un modelo general y solo es aplicable cuando los turnos de clase sean de duración constante. El segundo modelo es más general, pero como ha sido planteado no puede ser resuelto, para instancias reales, aplicando algoritmos clásicos. En esencia, se hace necesario contar con un modelo general que pueda ser resuelto en un tiempo computacional razonable. Para lograr esto el autor de este trabajo propone sacrificar la optimalidad y en su lugar se conforma con un modelo general y un algoritmo de solución que permita obtener "buenas soluciones" en un tiempo computacional razonable y con la utilización de pocos recursos. En la sección siguiente se plantea un modelo matemático heurístico general cuya formulación permite la utilización de algoritmos heurísticos/meta-heurísticos de una manera directa.

2.3. Modelo general heurístico

Las principales dificultades que, desde el punto de vista computacional, presenta el segundo modelo son:

1. Posee un número considerable de variables y restricciones.

2. No puede ser resuelto por algoritmos clásicos.

Para resolver la primera dificultad se regresa a la formulación propuesta para el primer modelo y se utilizan las restricciones planteadas, es decir se consideran:

Variables del modelo

X_{ij}: Cantidad de turnos de clase a impartir de la asignatura i en la semana j.
$i \in I$, $j \in J$

El número de variables de decisión es ahora de $n \times m$.

Restricciones del modelo

La cantidad de turnos de clase a impartir de la asignatura i durante la semana j tiene que estar entre la cantidad mínima y la máxima permitida.

$$lb_{ij} \leq X_{ij} \leq ub_{ij} \quad \forall i \in I \quad \text{y} \quad \forall j \in J \tag{2.17}$$

Se tiene como restricción para la cantidad de turnos de clase:

$$\sum_{j \in J} X_{ij} = p_i \quad \forall i \in I \tag{2.18}$$

Para considerar turnos de clase de duración arbitraria se plantea la siguiente fórmula que permite calcular la carga semanal considerando esta situación.

$$C_j = \sum_{i \in I} \left(\sum_{k=\sum_{s=1}^{j-1} X_{is}+1}^{\sum_{s=1}^{j} X_{is}} h_{ik} \right) \tag{2.19}$$

Para no exceder las horas disponibles en una semana se plantea:

$$C_j \leq Q_j \quad \forall j \in J \tag{2.20}$$

Función objetivo del modelo general heurístico

Como en los modelos anteriores, el objetivo es minimizar las desviaciones cuadradas de las cargas semanales respecto al promedio:

$$\text{mín} \sum_{j \in J} \left(C_j - \frac{1}{m} \sum_{j \in J} C_j \right)^2 \tag{2.21}$$

Alcance de la aplicación del modelo

Es obvio que debido a la naturaleza de la expresión 2.19 resulta imposible utilizar las restricciones 2.20 y 2.21 en el contexto de la programación lineal, de modo que tampoco pueden emplearse los algoritmos clásicos de la programación lineal descritos en la sección 1.3. No obstante, es necesario aclarar que este modelo es equivalente al modelo de la sección 2.2 en términos de generalidad y que una solución óptima obtenida con la formulación 2.3 es también una solución óptima para el modelo formulado en 2.2. Además, por la estructura propia del modelo resulta más conveniente utilizar un método heurístico/meta-heurístico bajo esta formulación que bajo aquella formulada en 2.2. Es por esta razón que el autor de este trabajo le da a la formulación obtenida en esta sección la denominación de modelo general heurístico.

Para minimizar 2.21 se utiliza TS, los resultados numéricos al emplear esta formulación se exponen en el capítulo tres. El problema fundamental que presenta esta formulación, que es totalmente aceptado por el autor de este trabajo, es que con la utilización del algoritmo de solución propuesto en la siguiente sección no puede asegurarse de manera absoluta que se obtiene un óptimo global del problema de balance de carga docente. De hecho resulta imposible reconocer cuándo se ha alcanzado un óptimo local/global debido a que, por las bases heurísticas del algoritmo, no se tiene ningún criterio de optimalidad, a diferencia del que presenta la programación lineal. El sacrificio de la optimalidad en aras de obtener "buenas soluciones" en un tiempo computacional razonable es de hecho una ventaja. Esta aseveración parece contradictoria pero deja de serlo si se observa de que de nada sirve un algoritmo que es conocido que conducirá al óptimo global pero que una vez alcanzado, esta solución ya no resulta práctica puesto que en el momento en el que se necesitó no estaba aún disponible.

2.4. Pseudocódigo de TS para el problema de balance de carga docente

En la sección 1.4.1 se presenta una descripción del algoritmo meta-heurístico de Búsqueda Tabú y su esquema general. En esta sección se definen, de manera particular para el modelo 2.3, cómo obtener una solución inicial para el problema de balance de carga docente, los movimientos que permiten explorar la vecindad de las soluciones candidatas y los movimientos tabú. La elección de esta meta-heurística sobre las demás analizadas se justifica porque permite aprovechar la estructura propia del problema que se intenta resolver de manera directa, sin el consumo de recursos computacionales adicional que presupone la utilización de algoritmos basados en inteligencia de enjambre. Como puede verse a continuación, el autor de este trabajo propone una metodología que garantiza la satisfacción de todas las restricciones de igualdad del modelo planteado (las restricciones 2.18) y las de caja (restricciones 2.17). En el caso de las desigualdades (restricciones de carga semanales 2.20), la metodología promueve la convergencia hacia un punto factible (posiblemente óptimo[1]) del problema. Esto está sustentado por el hecho de que la sucesión generada por el algoritmo TS aproxima la mejor solución obtenida a la región factible, obteniendo al mismo tiempo un mejor balance.

Para realizar un movimiento con el algoritmo propuesto se seleccionan aleatoriamente una asignatura i y dos semanas s_1 y s_2, $s_1 \neq s_2$ y se efectúa: $X_{is_1} = X_{is_1} + 1$ y $X_{is_2} = X_{is_2} - 1$. Esta operación, obviamente, no viola la restricción dada en 2.18, pero no asegura que las restricciones de caja 2.17 y las restricciones de carga semanal dadas en 2.20 no sean violadas. Más adelante se muestra cómo manejar estas restricciones; primero es conveniente explicar el procedimiento que se utiliza para generar una solución inicial que no viola las restricciones dadas en 2.17 y 2.18:

[1]Los resultados experimentales la sección 3.3 sugieren que en general se converge a un óptimo global.

1. Inicializar en cero todas las variables.

2. Asignar a cada asignatura, mientras se disponga de turnos de clase, su cota inferior en cada semana del período.

3. Para cada asignatura, aumentar en uno el número de encuentros a impartir en cada semana del período, siempre y cuando no se excedan su cota superior y la carga máxima permitida en esa semana.

4. Volver al paso 3 mientras queden turnos de clase por asignar y éstos puedan ser asignados sin violar las restricciones 2.17 y 2.20.

5. Si aún restan turnos de clase por asignar:

   a) Determinar las semanas donde se dispone del máximo de horas clases.

   b) Para cada asignatura, repartir sus turnos de clase restantes entre las semanas donde se dispone del máximo de horas, excepto en aquellas semanas en las que la asignación viole la cota superior en la restricción 2.17.

   c) Si aún quedan turnos de clase por asignar, informar al usuario de que muy probablemente se trata de un problema infactible y que debe aumentar la cota superior para la cantidad de turnos de clase a impartir de las asignaturas para las cuales sus turnos de clase no pudieron ser totalmente asignados.

La solución inicial que resulta de este procedimiento es, por lo general, infactible debido a que viola algunas restricciones dadas en 2.20. Para manejar estas restricciones puede utilizarse el método de las penalizaciones, sin embargo este método de manejo de restricciones no es necesario ya que estas penalizaciones están incluidas en la propia función objetivo.

A continuación se formalizan las condiciones bajo las cuales los movimientos son considerados válidos o admisibles:

Movimiento admisible

A partir de una solución X se obtiene otra solución $X^{'}$ aplicando el movimiento $X_{is_1} = X_{is_1} + 1 \ \wedge \ X_{is_2} = X_{is_2} - 1$ si se satisfacen las condiciones siguientes:

1. El movimiento es realizado en semanas distintas, $s_1 \neq s_2$.
2. El movimiento no está en la lista tabú.
3. El resultado de la operación $X_{is_2} = X_{is_2} - 1$ tiene que satisfacer $X_{is_2} \geq lb_{is_2}$.
4. El resultado de la operación $X_{is_1} = X_{is_1} + 1$ tiene que satisfacer $X_{is_1} \leq ub_{is_1}$.
5. No se violan las restricciones 2.20 en las semanas $\widetilde{J} = \{s_1, s_1 + 1, \ldots, s_2 - 1\} - J^{max}$ si $s_1 < s_2$, o bien, $\widetilde{J} = \{s_2 + 1, s_2 + 2, \ldots, s_1\} - J^{max}$ si $s_1 > s_2$.

Para entender mejor el efecto de estos movimientos y la necesidad de solo analizar la satisfacción de las restricciones 2.20 en las semanas $\{s_1, s_1 + 1, \ldots, s_2 - 1\} - J^{max}$ si $s_1 < s_2$, o bien, $\{s_2 + 1, s_2 + 2, \ldots, s_1\} - J^{max}$ si $s_1 > s_2$, considérese una asignatura hipotética con doce turnos de clase a impartir en un período de cinco semanas.

Tabla 2.1: Efecto de los movimientos definidos para el algoritmo propuesto (Caso $s_1 < s_2$).

	s1		**s2**	
1	3	4	2	2
(1-1)	(2-4)	(5-8)	(9-10)	(11-12)
1	4	4	1	2
(1-1)	(2-5)	(6-9)	(10-10)	(11-12)

Tabla 2.2: Efecto de los movimientos definidos para el algoritmo propuesto (Caso $s_1 > s_2$).

	s2		**s1**	
1	3	4	2	2
(1-1)	(2-4)	(5-8)	(9-10)	(11-12)
1	2	4	3	2
(1-1)	(2-3)	(4-7)	(8-10)	(11-12)

En la tabla 2.1 puede observarse que al realizar un movimiento en las semanas s_1 y s_2 tomando $s_1 < s_2$, la cantidad de turnos de clase a impartir de esta asignatura en las semanas posteriores a s_1 y anteriores a s_2 no cambia, aunque sí lo hace en calidad de orden, es decir, se corren los turnos de clase hacia la derecha. Esto evidentemente

puede cambiar la carga semanal, aumentándola o disminuyéndola en estas semanas, en dependencia de la duración de estos turnos que fueron corridos. Es también evidente que la carga en la semana s_1 aumentará y en la semana s_2 disminuirá, ya que se incrementó y decrementó en uno la cantidad de turnos de clase de esta asignatura en las semanas s_1 y s_2 respectivamente. Tienen que verificarse entonces las restricciones 2.20 en las semanas s_1, s_1+1 hasta s_2-1, pues ya es conocido que en la semana s_2 la carga disminuye. Se excluyen del análisis las semanas del conjunto $\{s_1, s_1+1, \ldots, s_2-1\}$ que pertenezcan además al conjunto J^{max}, pues por construcción, la solución inicial ya las viola y el algoritmo TS al minimizar la función objetivo del modelo 2.3 promueve aquellas soluciones que se acercan a la región factible.

En la tabla 2.2 puede observarse que al realizar un movimiento en las semanas s_1 y s_2 tomando $s_1 > s_2$, la cantidad de turnos de clase a impartir de esta asignatura en las semanas posteriores a s_2 y anteriores a s_1 no cambia, aunque al igual que en el caso anterior sí lo hace en calidad de orden, es decir, ahora se corren los turnos de clase hacia la izquierda. Esto, como ya ha sido explicado, puede cambiar la carga semanal, aumentándola o disminuyéndola en estas semanas, en dependencia de la duración de los turnos que fueron corridos. Resulta también evidente que la carga en la semana s_1 aumentará y en la semana s_2 disminuirá, ya que se incrementó y decrementó en uno la cantidad de turnos de clase de esta asignatura en las semanas s_1 y s_2 respectivamente. En este caso tienen que verificarse las restricciones 2.20 en las semanas s_2+1, s_2+2 hasta s_1, pues ya es conocido que en la semana s_2 la carga disminuye. Se excluyen del análisis las semanas del conjunto $\{s_2+1, s_2+2, \ldots, s_1\}$ que pertenezcan además al conjunto J^{max}, pues como ha sido explicado, el algoritmo propuesto para la obtención de una solución inicial tiene explícito en su diseño la violación de estas restricciones de carga en las semanas del conjunto J^{max}.

Como resultado se plantea el siguiente algoritmo en pseudocódigo que es el esquema básico a seguir para la programación de la solución inicial.

Algoritmo B: Solución inicial.

I, J, F, c, p, Q;

Output: X_0;

$X := 0$;

$ecuentros_i := p_i$;

foreach $i \in I$ **do**

  foreach $j \in J$ **do**

    $X_{ij} := lb_{ij}$;

    $encuentros_i := encuentros_i - lb_{ij}$;

  end

end

$totalencuentros = \sum_{\forall i \in I} encuentros_i$;

while $totalencuentros \neq 0$ **do**

  $l := totalencuentros$;

  foreach $i \in I$ **do**

    foreach $j \in J$ **do**

      if $encuentros_i > 0$ **and** $X_{ij} < ub_{ij}$ **and** $C_j \leq Q_j | (X_{ij} + 1)$ **then**

        $X_{ij} := X_{ij} + 1$;

        $encuentros := encuentros - 1$;

        $totalencuentros := totalencuentros - 1$;

        if $encuentros_i = 0$ **then**

          exitloop;

        end

      end

    end

  end

  if $l = totalencuentros$ **then**

    exitloop

  end

end

Algoritmo B: Solución inicial (Continuación).

```
if totalencuentros ≠ 0 then
    J^max:=indexesOf(máx Q_j);
    foreach i ∈ J do
        while encuentros_i ≠ 0 do
            l = encuentros_i;
            foreach j ∈ J^max do
                if X_ij < ub_ij then
                    X_ij := X_ij + 1;
                    encuentros_i := encuentros_i − 1;
                    if encuentros = 0 then
                        exitloop
                    end
                end
            end
            if l = encuentros_i then
                error No pueden asignarse todos los encuentros de la asignatura i.
                Considere aumentar la cota superior para los encuentros semanales de esta
                asignatura.
            end
        end
    end
end
```

A continuación se define lo que se considera movimiento tabú en el algoritmo.

Movimiento tabú

Si sobre una solución X se efectúa un movimiento admisible (no viola restricciones o acerca una solución infactible a la región de factibilidad cumpliendo con lo descrito anteriormente) dando como resultado $X^{'}$, se considera tabú al movimiento inverso, o sea el movimiento que de ser efectuado sobre $X^{'}$ da como resultado X. La idea de colocar estos movimientos en una lista e impedir la utilización de éstos durante una cierta cantidad de iteraciones resulta ventajosa para prevenir la ejecución de ciclos en el algoritmo. Por ejemplo la figura 2.1 muestra una posible sucesión del algoritmo TS. En este ejemplo la mejor solución en una vecindad de la solución actual es escogida como reemplazo de la solución actual siempre y que mejore su evaluación en la función objetivo. La secuencia converge a un posible

óptimo del problema en un número finito de iteraciones. Sin un mecanismo diseñado para prevenir ciclos, la secuencia pudo haber sido la mostrada en la figura 2.2. En el mejor de los casos el posible óptimo del problema es alcanzado en más iteraciones (considerando que dado el carácter estocástico del algoritmo, éste pudiera escapar del ciclo.). En el peor de los casos la sucesión generada queda atrapada en el ciclo consumiendo así todas las iteraciones, la cantidad de evaluaciones de la función objetivo predeterminada o cualquier otra condición de terminación establecida por el investigador.

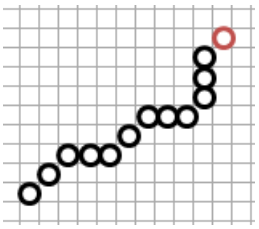


Figura 2.1: Una posible sucesión generada por TS.

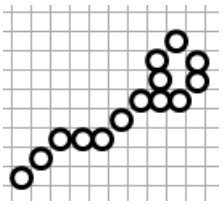


Figura 2.2: Una posible sucesión generada por un algoritmo que explora vecindades pero no instrumenta la prevención de ciclos.

Vecindad admisible

Una vecindad admisible de X es un conjunto finito de soluciones candidatas obtenidas a través de movimientos admisibles. La vecindad admisible de una solución no incluye a la solución en sí.

Una vecindad admisible de una solución se obtiene generando N movimientos admisibles. Este procedimiento se resume con el siguiente algoritmo en pseudocódigo:

Algoritmo C: Vecindad admisible de tamaño N de la solución S.

N, S, T, I, J, Q, M;
Output: $\widetilde{N}(S)$
Inicialización;
$\widetilde{N}(S) := \emptyset$;
$contador := 0$;
while $N \neq 0$ **and** $contador \neq M$ **do**
    $i :=$ `RandomSample`$(I,\ 1)$;
    $s_1 :=$ `RandomSample`$(J,\ 1)$;
    $s_2 :=$ `RandomSample`$(J - \{s_1\},\ 1)$;
    if $(i,\ s_1,\ s_2) \notin T$ **and** $X_{is_2} - 1 \geq lb_{is_2}$ **and** $X_{is_1} + 1 \leq ub_{is_1}$ **and**
    $C_j \leq Q_j | (X_{is_1} + 1, X_{is_2} - 1) \forall j \in \widetilde{J}$ **then**
        $X :=$ `Copy`(S);
        $X_{is_1} := X_{is_1} + 1$;
        $X_{is_2} := X_{is_2} - 1$;
        $\widetilde{N}(S) := \widetilde{N}(S) \bigcup \{(X,\ (i,\ s_2,\ s_1))\}$;
        $N := N - 1$;
    end
    $contador := contador + 1$;
end

El parámetro M en el pseudocódigo anterior garantiza que el ciclo se detenga aún cuando no se haya podido determinar una vecindad admisible de S. Si es el caso, entonces se considera que el problema no tiene solución.

2.5. Conclusiones del capítulo

1. Se formula el modelo 2.1 que considera turnos de clase de duración constante. Es además un modelo que puede ser resuelto por algoritmos clásicos para la optimización en enteros puros.

2. Se formula el modelo 2.2 para turnos de clase de duración arbitraria. Un modelo más general que incorpora todos los requerimientos identificados en el capítulo uno pero que no puede ser resuelto para instancias reales del problema de balance de carga docente.

3. La formulación 2.3 (un modelo general equivalente al formulado en 2.2) y la utilización del método meta-heurístico TS, reduce significativamente el número de variables y restricciones en comparación con los modelos de PLE y garantiza la obtención de una buena solución en un tiempo computacional razonable.

4. Se plantean los algoritmos en pseudocódigo para obtener una solución inicial del problema de balance de carga docente y la exploración de la vecindad de las soluciones candidatas. Estos algoritmos en pseudocódigo pueden ser implementados en cualquier lenguaje de programación.

Capítulo 3

Experimentación numérica y Automatización

Introducción

En las secciones siguientes son aplicadas pruebas estadísticas a los resultados numéricos del modelo heurístico 2.3 y el algoritmo de solución 2.4 planteados en el capítulo dos. Estas pruebas corroboran la hipótesis de esta investigación. Además, se selecciona un lenguaje de programación y sus respectivas bibliotecas numéricas para la solución automatizada del problema de balance de carga docente. Como resultado se obtiene una aplicación de escritorio denominada *QBalance* que asiste a los planificadores en la confección de los balances de carga docente.

3.1. Pruebas para la comparación de dos muestras relacionadas

La necesidad de comparar dos muestras relacionadas es muy frecuente en la práctica científica cuando, por ejemplo, se está probando la eficacia de un determinado tratamiento. En este caso, se trata de comparar dos procedimientos para la confección de balances de carga docente, el procedimiento manual y el procedimiento que utiliza el modelo y el algoritmo propuestos por el autor de este trabajo, teniendo en cuenta el tiempo para completar la confección de un balance de carga docente (tiempo de planificación), la cantidad de errores cometidos y por último la calidad del balance de carga obtenido que resulta medida por la función objetivo del modelo de la sección 2.3.

La literatura reporta fundamentalmente la utilización de dos pruebas: la prueba t de student, basada en el estadístico t introducido en el año 1908 por William Sealy Gosset (Student, 1908) quien firmaba sus trabajos con el seudónimo "Student", para muestras dependientes pareadas y la prueba de rangos con signos de Wilcoxon (Wilcoxon, 1945), publicada en el año 1945 por Frank Wilcoxon.

El estadístico t de la prueba t de student para la comparación de dos muestras relacionadas tiene por ecuación:

$$t = \frac{\bar{X}_d - \mu_0}{s_d/\sqrt{n}}$$

se utiliza para probar si el promedio de las diferencias es significativamente distinto de μ_0. En esta ecuación, $\bar{X}_d$, para este caso, resulta ser el promedio de las diferencias entre ambos procedimientos tomando como indicador una de las variables: "Tiempo de planificación", "Errores del balance" o "evaluación de la función objetivo" del modelo de la sección 2.3, s_d la desviación estándar de la muestra, n el tamaño de la muestra y por último $\mu_0 = 0$, ya que la hipótesis nula para esta prueba es que no existen diferencias significativas entre los valores promedio de la variable en cuestión entre ambos procedimientos.

Los supuestos al aplicar la prueba t de student para la comparación de dos muestras relacionadas son:

1. La variable dependiente debe medirse en una escala continua, es decir, que se mide en el intervalo o nivel de razón. Ejemplos de variables que satisfacen este criterio incluyen: el tiempo de revisión (medido en horas), la inteligencia (medida mediante puntuación de CI), el rendimiento de un examen (medido de 0 a 100), peso (medido en kg).

2. La variable independiente debe constar de dos "grupos relacionados" o "pares" categóricos. "Grupos relacionados" indica que los mismos sujetos están presentes en ambos grupos. La razón por la cual resulta posible tener los mismos sujetos en cada grupo se debe a que cada sujeto ha sido medido en dos ocasiones para la misma variable dependiente.

3. La distribución probabilística de las diferencias es normal.

4. La desviación estándar de esta distribución es desconocida.

Para las situaciones que involucran elementos coincidentes o mediciones repetidas del mismo caso, se puede utilizar la prueba no paramétrica de rangos con signos de Wilcoxon. Esta prueba se utiliza por lo general cuando los supuestos de la prueba t no se cumplen satisfactoriamente. Cuando se violan los supuestos de la prueba t es muy probable que la prueba de rangos con signos de Wilcoxon proporcione buenos resultados en la detección de diferencias significativas, ya que tiene menos supuestos que la prueba t. Por otra parte, incluso en condiciones apropiadas para la aplicación de la prueba t, la prueba de rangos con signos de Wilcoxon ha demostrado ser casi tan poderosa.

Los supuestos para la prueba de rangos con signos de Wilcoxon son:

1. La variable dependiente debe medirse en el nivel ordinal o continua. Ejemplos de variables ordinales incluyen escalas de Likert (por ejemplo, una escala de 7 puntos desde "muy de acuerdo" hasta "muy en desacuerdo"), entre otras formas de categorías de clasificación (por ejemplo, una escala de 5 puntos que explica lo mucho que a un cliente le gusta un producto, que van desde "No mucho" a "Sí, mucho").

2. La variable independiente debe constar de dos "grupos relacionados" o "pares" categóricos.

3. La distribución de las diferencias entre los dos grupos relacionados (es decir, la distribución de las diferencias entre las puntuaciones de los dos grupos de la variable independiente, por ejemplo, el tiempo de reacción en una habitación con "luz azul" y una habitación con "luz roja") tiene que ser simétrica. Si la distribución de las diferencias tiene forma simétrica, entonces se pueden analizar los datos experimentales mediante la prueba de rangos con signo de Wilcoxon.

Para realizar la prueba de rangos con signos para detectar diferencias significativa entre las medianas, primero se obtiene el estadístico W de esta prueba.

1. Para cada caso en la muestra de n casos, calcular la diferencia D_i entre las observaciones.

2. Omitir el signo de la diferencia para obtener un conjunto de n diferencias absolutas $\mid D_i \mid$.

3. Omitir para el análisis posterior cualquier diferencia absoluta con valor cero. De este modo se obtiene un nuevo conjunto de $\hat{n}$ de diferencias absolutas distintas de cero,

donde $\hat{n} \leq n$. Nótese que $\hat{n}$ es el tamaño real de la muestra, luego de haber extraído las diferencias con valor absoluto igual a cero.

4. Asignar los rangos R_i de 1 hasta $\hat{n}$ a cada una de las diferencias absolutas $\mid D_i \mid$, de modo que a la diferencia absoluta más pequeña le sea asignado el rango 1 y a la diferencia absoluta mayor le sea asignado el rango $\hat{n}$. Debido a la falta de precisión en el proceso de medición, si dos o más diferencias absolutas son iguales, entonces a estas diferencias absolutas se les asigna el promedio de los rangos que les serían asignados individualmente.

5. Reasignar los signos a cada uno de los $\hat{n}$ rangos R_i, dependiendo del signo que originalmente tenía D_i.

6. El estadístico W de Wilcoxon se calcula sumando todos los rangos positivos.

$$W = \sum_{i=1}^{\hat{n}} R_i^{(+)}$$

3.2. Resultados numéricos del modelo heurístico

Para la validación del modelo heurístico formulado en la sección 2.3 se tomó una muestra aleatoria de catorce balances de carga docente de los últimos cursos. En la tabla 3.1, la primera columna contiene los resultados de evaluar la solución proporcionada por los planificadores docentes en la función objetivo del modelo 2.3 y la segunda columna contiene los resultados de esta función objetivo cuando se resuelven los catorce balances mediante el modelo y el algoritmo propuestos. En la tabla 3.2 se muestran las diferencias en la función objetivo cuando se emplea el procedimiento manual y el procedimiento automatizado. Se procede a comprobar que existen diferencias en los valores de la función objetivo de ambos procedimientos y que en efecto se obtienen balances de carga docente mejores cuando se utilizan el modelo y el algoritmo propuestos.

Con este fin se emplea la prueba t de student de la sección 3.1. Inicialmente se comprueban los supuestos para esta prueba.

1. La variable dependiente debe medirse en una escala continua.

   Este supuesto es fácilmente verificable pues el resultado de evaluar la solución obtenida manualmente por los planificadores docentes y la solución del modelo y el algoritmo propuestos en la función objetivo del modelo de la sección 2.3 es un número real que pertenece al intervalo $[0,\ +\infty)$. De modo que la diferencia entre el procedimiento manual y el procedimiento automatizado propuesto es un número real que pertenece al intervalo $(-\infty,\ +\infty)$. Se trata entonces de una variable que es medida en una escala continua.

2. La variable independiente debe constar de dos "grupos relacionados" o "pares" categóricos.

   Los datos presentados en la tabla 3.1 muestran para cada balance de carga docente: las evaluaciones en la función objetivo del modelo propuesto en la sección 2.3 de la solución manual obtenida por los planificadores docentes y de la solución obtenida por el procedimiento automatizado.

Tabla 3.1: Muestras obtenidas en la experimentación numérica

Función Objetivo (Balance Manual)	**Función Objetivo (Modelo propuesto)**
69,33333333	9,6
301	16
146,6666667	3,666666667
46,4	6,4
436,4375	9,4375
421	8,9375
118,9333333	8,933333333
304,3076923	4,307692308
12,4375	4,4375
64	0
196,8571429	6,857142857
1002,857143	36,85714286
138,8571429	7,714285714
225,2307692	4

Tabla 3.2: Diferencias en la función objetivo entre el procedimiento manual y el automatizado

Manual-Automatizado
59,7333333
285
143
40
427
412,0625
110
300
8
64
190
966
131,142857
221,230769

3. La distribución probabilística de las diferencias es normal. El histograma mostrado en la figura 3.1 no resulta concluyente acerca de si la distribución de las diferencias sigue en efecto una distribución normal.

   Para comprobar si en efecto la variable en estudio puede ajustarse adecuadamente a una distribución normal se aplica la prueba no paramétrica de Kolmogorov-Smirnov. Los resultados de esta prueba se muestran en la tabla 3.3. Como la significación asintótica de la prueba no paramétrica de Kolmogorov-Smirnov resultó ser mayor que 0.05 no puede rechazarse la hipótesis de que las diferencias observadas en la función objetivo del modelo de la sección 2.3 entre el procedimiento manual y el automatizado provienen de una distribución normal con un 95 % de confianza.

4. La desviación estándar de esta distribución es desconocida.

   La desviación estándar de la población es desconocida. Si se conociese, la prueba estaría basada en el estadístico Z cuya distribución es $N(0,\ 1)$.

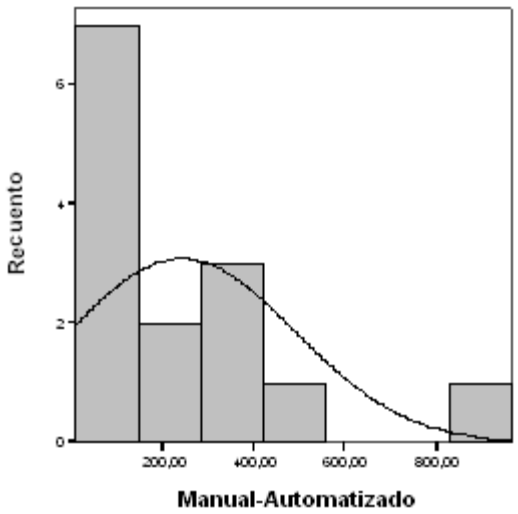


Figura 3.1: Histograma de las diferencias entre los resultados del procedimiento manual y el automatizado.

Tabla 3.3: Prueba de Kolmogorov-Smirnov para una muestra (Distribución de contraste *Normal*)

		Manual-Automatizado
	N	14
Parámetros normales	Media	239,7978
	Desviación típica	247,38356
Diferencias más extremas	Absoluta	0,19
	Positiva	0,19
	Negativa	-0,174
Z de Kolmogorov-Smirnov		0,709
Sig. asintót. (bilateral)		0,696

El autor de este trabajo ha verificado los supuestos para la prueba t. Se procede a comprobar, empleando esta prueba, que existen diferencias significativas entre ambos procedimientos.

Hipótesis

H_0**:** No existen diferencias en los valores de la función objetivo entre ambos procedimientos.

H_1**:** Existen diferencias en los valores de la función objetivo entre ambos procedimientos.

Tabla 3.4: Resultados de la Prueba de t para muestras relacionadas

Media	Desviación típ.	95 % Intervalo de confianza		t	Sig. Bilateral
		Superior	Inferior		
239,79782	247,38356	96,9628	382,63284	3,627	0,003

Como se observa en la tabla 3.4, debido a que el p-value $(0,003)$ para esta prueba es menor que $0,05$ se rechaza la hipótesis nula con un 95 % de confianza. Es decir, el resultado de la prueba indica que existen diferencias en los valores de la función objetivo entre ambos procedimientos. Además de confirmar la existencia de diferencias entre ambos procedimientos, el promedio de las diferencias resulta ser positivo, lo cual indica que los valores de la función objetivo son menores (se obtiene un mejor equilibrio de horas clases semanales durante el período de planificación) cuando se utilizan el modelo y el algoritmo propuestos.

No se procede a emplear pruebas de hipótesis para confirmar que las variables "Tiempo de planificación" y "Errores del balance de carga docente" mejoran al utilizar el modelo y el algoritmo propuestos ya que al emplear el software que automatiza este procedimiento se completa un balance en aproximadamente 40 segundos a diferencia de la planificación manual que requiere dos o tres días. Por otro lado, el balance de carga docente resultante satisface todas las restricciones del modelo propuesto (no se cometen errores) y en el caso de la planificación manual los planificadores reportan un promedio de 4 errores por balance.

3.3. Análisis de convergencia del algoritmo propuesto

Al utilizar un algoritmo heurístico/meta-heurístico para la solución de problemas de optimización es importante tener una idea acerca de cuán buena es la solución obtenida. Si se dispone, como es el caso, de soluciones óptimas para distintas instancias, entonces puede aplicarse la prueba t para muestras relacionadas de la sección 3.1 o la prueba de rangos con signos de Wilcoxon también la sección 3.1 si no se cumplen los supuestos para la prueba t, con el objetivo de determinar si existen diferencias entre los valores respectivos de la función objetivo en las soluciones óptimas conocidas y aquellos valores de la función objetivo en las soluciones obtenidas por el algoritmo heurístico/meta-heurístico. Para comprobar si el algoritmo propuesto converge en general hacia una solución óptima global, el autor de este trabajo tomó una muestra aleatoria de quince instancias en las que la evaluación óptima de la función objetivo es conocida. Estas instancias fueron resueltas utilizando el algoritmo propuesto, los resultados se muestran en la tabla 3.5. Las instancias pertenecen a balances de carga docente en los que la duración de cada turno de clase es constante e igual a dos horas (modelo de la sección 2.1). No se incluyeron instancias del modelo con turnos de duración arbitraria 2.2, pues como se planteó en el capítulo uno estas instancias constituyen problemas NP- duro y no fue posible resolverlas en un tiempo computacional aceptable.

El autor de este trabajo decide no realizar una prueba de hipótesis para confirmar la convergencia del algoritmo propuesto y efectuar solo un análisis de los datos tabulados pues no existe evidencia de variabilidad en los datos. Obsérvese que los valores numéricos presentados en la tabla 3.5 coinciden en ambas columnas excepto en el cuarto caso. De modo que no es posible aplicar ninguna de las pruebas de la sección 3.1 puesto que para el caso de la prueba t para muestras relacionadas la hipótesis de normalidad claramente no se cumple y en el caso de la prueba de rangos con signos de Wilcoxon solo quedaría un caso, o sea el tamaño de la muestra real sería $\hat{n} = 1$ luego de haber eliminado del conjunto de datos las diferencias con valor absoluto cero. Es por esta razón que el autor de este trabajo solo realiza un análisis descriptivo de los datos presentados en la tabla 3.5.

De la tabla 3.5 se concluye que el algoritmo propuesto ha alcanzado un óptimo global en 14 de las 15 instancias resueltas para un 93 %. Debe señalarse además que, no necesariamente las soluciones óptimas de ambos procedimientos tienen que ser iguales puesto que el hecho de obtenerse distintas soluciones con igual evaluación óptima confirma la existencia de múltiples óptimos globales. Esta observación es importante ya que los planificadores

pueden analizar otras alternativas simplemente volviendo a resolver el problema y debido al carácter estocástico del algoritmo propuesto, éste conduciría generalmente a otra solución óptima. En el caso de turnos de clase de duración arbitraria aunque no se tienen instancias para efectuar una comparación similar a la que se realizó con los datos de la tabla 3.5, sí se tiene una idea del comportamiento del algoritmo propuesto en esta situación, pues como se muestra en el anexo D, los balances obtenidos para seis instancias presentan un buen equilibrio de horas clases semanales.

Tabla 3.5: Muestras obtenidas en la experimentación numérica con el solucionador exacto y el algoritmo propuesto

Función Objetivo (Solucionador Exacto)	**Función Objetivo (Algoritmo propuesto)**
16	16
7,157894737	7,157894737
14,4	14,4
7,15789473684	18,52631579
10,66666667	10,66666667
12,92307692	12,92307692
0	0
11,42857143	11,42857143
0	0
14,4	14,4
0	0
14,93333333	14,93333333
13,71428571	13,71428571
16	16
0	0

3.4. Elección de un lenguaje de programación y las bibliotecas numéricas

Python es un lenguaje de programación potente y fácil de aprender. Cuenta con estructuras de datos de alto nivel eficientes y un enfoque simple pero eficaz para programación orientada a objetos[1]. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para secuencias de comandos y el desarrollo rápido de aplicaciones en muchas áreas en la mayoría de las plataformas.

El autor de este trabajo tiene ya varios años de experiencia con este lenguaje de programación y lo ha seleccionado para otros proyectos de investigación. Es por esta razón principalmente y por las ventajas que Python ofrece que el autor considera su utilización en la implementación del software para la confección de balances de carga docente.

Para la automatización del algoritmo de solución se utilizan fundamentalmente las bibliotecas de Python: PulP (para la solución exacta del modelo 2.7), PyBrain (para la codificación del algoritmo Búsqueda Tabú para el balance de carga docente) y PyQt (para la codificación de la interfaz gráfica de usuario). En las secciones siguientes se describen brevemente estas bibliotecas y se dejan referencias para el lector interesado en el tema.

PulP es una biblioteca para el lenguaje de programación Python que permite a los usuarios describir modelos matemáticos. PulP funciona completamente dentro de la sintaxis y expresiones naturales del lenguaje Python, proporcionando objetos de Python que representan problemas de optimización, variables de decisión y restricciones que son expresados de una manera muy similar a las expresiones matemáticas originales. Para mantener la sintaxis tan simple e intuitiva como sea posible, PulP se ha centrado en los modelos lineales mixtos y enteros. PulP puede ser fácilmente desplegado en cualquier sistema que tenga un intérprete de Python, ya que no tiene dependencias con otros paquetes de software. Es compatible con una amplia gama de solucionadores tanto comerciales como de código abierto, y puede ser fácilmente extendido para admitir solucionadores adicionales. En el diseño de PulP se consideró que resultaba conveniente que fuese utilizable en cualquier lugar, ya sea para ser utilizado directamente como un modelador y herramienta de experimentación, o como parte de una aplicación más

[1]El autor de este trabajo utiliza expresiones como: programación orientada a objetos, clase, método, función, entre otras relativas al diseño de sistemas informáticos. No se ofrecen definiciones formales o informales de estos conceptos pues no se considera pertinente. El lector interesado puede tener una introducción al tema en cualquier libro de ingeniería de software.

considerable. Esto requiere que PulP sea asequible, y adaptable a diferentes entornos de hardware y software. Muchos solucionadores para la programación lineal entera mixta están disponibles y pueden ser invocados desde PulP. Como la interfaz para muchos solucionadores es similar, o puede ser manejada traduciendo el modelo a los formatos estándar LP o MPS, se incluyen clases base genéricas, además de interfaces específicas para los solucionadores actualmente populares. Estas clases para los solucionadores genéricos pueden ser ampliadas por los usuarios o los desarrolladores de nuevos solucionadores con el mínimo esfuerzo.(Mitchell, O'Sullivan, y Dunning, 2011). En la experimentación numérica realizada por el autor de este trabajo se utilizó el solucionador coinMP[2], incluido por defecto en la instalación de PulP.

PyBrain es una biblioteca de aprendizaje automático modular para Python. Su objetivo es ofrecer algoritmos flexibles y fáciles de usar pero potentes para tareas de aprendizaje automático y una variedad de entornos predefinidos para probar y comparar algoritmos. PyBrain implementa muchos algoritmos de aprendizaje recientes y arquitecturas que van desde temas como el aprendizaje supervisado y aprendizaje por refuerzo, la búsqueda directa, optimización y algoritmos evolutivos. Su única dependencia estricta es la biblioteca científica para Python SciPy. Como es típico para la programación en Python /SciPy, el tiempo de desarrollo se reduce en gran medida en comparación con lenguajes como Java/C++, a costa de una velocidad más baja. Esta biblioteca va más allá de las bibliotecas de Python existentes, ya que proporciona un conjunto de herramientas para la el aprendizaje supervisado, no supervisado y por refuerzo, así para la optimización por caja negra y la optimización multi-objetivo.(Schaul y cols., 2010)

De esta biblioteca se utilizó básicamente su módulo de optimización **pybrain.optimization.optimizer**, y específicamente de este módulo la clase **BlackBoxOptimizer**. Esta clase ofrece una estructura general para la programación de algoritmos de optimización.

Para el diseño de la interfaz gráfica de usuario se utilizó PyQt. Ésta es una biblioteca multiplataforma de Python. Resulta una de las opciones de Python para la programación de interfaces gráficas de usuarios. PyQt es desarrollado por la firma británica Riverbank Computing. Esta biblioteca implementa más de 620 clases y más de 6000 funciones y métodos (Riverbankcomputing, 2014).

[2]coinMP resuelve problemas de programación lineal en enteros. Utiliza, entre otras, las relajaciones comentadas en la sección 1.3

Debido a que hay una gran cantidad de clases disponibles se han dividido en varios módulos:

- QtCore
- QtGui
- QtNetwork
- QtXml
- QtSvg
- QtOpenGL
- QtSql

El módulo QtCore contiene la funcionalidades que no son gráficas. Este módulo se utiliza para trabajar con el tiempo, los archivos y directorios, distintos tipos de datos, urls, tipos MIME, hilos o procesos. El módulo QtGui contiene los componentes gráficos y las clases relacionadas. Estos incluyen, por ejemplo, botones, ventanas, barras de estado, barras de herramientas, barras de desplazamiento, mapas de bits, colores, fuentes, etc. El módulo QtNetwork contiene las clases para la programación en red. Estas clases facilitan la codificación de los clientes y servidores TCP/IP y UDP[3] que hacen la programación en red más fácil y más portátil. El módulo QtXml contiene clases para trabajar con archivos en el formato XML. El módulo QtSvg ofrece clases para mostrar el contenido de archivos SVG[4]. El módulo QtOpenGL se utiliza para la representación en dos y tres dimensiones utilizando la librería OpenGL. El módulo QtSql ofrece clases para trabajar con sistemas bases de datos.

En el diseño de la interfaz gráfica de usuario el autor de este trabajo persiguió como objetivos: crear un ambiente donde el planificador docente pudiese manejar los términos y procedimientos que naturalmente utiliza cuando realiza balances manuales y presentar la respuesta del software ajustada lo mejor posible a estos términos. Véanse los anexos B y C.

[3]TCP, IP y UDP son protocolos para la comunicación y transmisión de paquetes en Internet.

[4]Scalable Vector Graphics (SVG) es un lenguaje para describir gráficos de dos dimensiones y aplicaciones gráficas.

3.5. Codificación de Búsqueda Tabú en Python[5]

Para la implementación de TS en Python se utiliza de la biblioteca PyBrain la clase **BlackBoxOptimizer**. En la implementación se redefine el método **__learnStep** que implementa un único paso del algoritmo. La lógica general es manejada internamente por la clase. Esto permite al desarrollador un alto nivel de abstracción.

A continuación se muestra la implementación en Python del esquema básico de TS de la sección 1.4.1.

```
from pybrain.optimization.optimizer import BlackBoxOptimizer
class TabuSearch(BlackBoxOptimizer):
    maxTabu, tabuList, bestSoFar = 20, [], None
    def _additionalInit(self):
        self.bestSoFar = self._initEvaluable
        self.bestSoFar.fitness = self._oneEvaluation(self._initEvaluable)
    def best(self, candidates):
        picker = min if self.minimize else max
        return picker(candidates, key = lambda p: p[0].fitness)
    def isThisOneBetter(self, candidate):
        if self.minimize:
            return candidate.fitness < self.bestSoFar.fitness
        return candidate.fitness > self.bestSoFar.fitness
    def _learnStep(self):
        candidateList = []
        for candidate, tabuMove in self.moves(self.bestSoFar):
            candidate.fitness = self._oneEvaluation(candidate)
            candidateList.append((candidate, tabuMove,))
        if candidateList:
            bestCandidate, tabuMove = self.best(candidateList)
            if self.isThisOneBetter(bestCandidate):
                self.bestSoFar = bestCandidate
                self.tabuList.insert(0, tabuMove)
                if len(self.tabuList) > self.maxTabu:
                    self.tabuList.pop()
```

Código en Python 3.1: Implementación de Búsqueda Tabú (tabu.py)

[5]Los códigos de Python que se muestran en esta sección son completamente funcionales y se utilizan en la experimentación numérica. Los resultados son expuestos en la sección 3.2.

Nótese que para esta implementación, en aras de lograr un mayor eficiencia en el código Python, existen diferencias en la lógica propia de algunos métodos con respecto a los pseudocódigos propuestos en la sección 2.4. El código que se muestra a continuación es una implementación en Python del algoritmo en pseudocódigo para la solución inicial de la sección 2.4.

```
# -*- coding: cp1252 -*-
from Solution import Solution
from scipy import zeros
def week_load(x, j, n, h):
        load = 0
        for i in range(n):
            sup = 0
            for s in range(j+1):
                sup += x[i][s]
            inf = 0
            for s in range(j):
                inf += x[i][s]
            for k in range(inf, sup):
                load += h[i][k]
        return load
def initial(n, m, p, Q, h, lbound = None, ubound = None):
    """ Obtención de una solución inicial """
    if not lbound: lbound = [[0]*m]*n
    if not ubound: ubound = [[6]*m]*n
    x = zeros((n, m,), int).tolist()
    meetings = p
    for i in range(n):
        for j in range(m):
            x[i][j] = lbound[i][j]
            meetings[i] -= lbound[i][j]
    totalmeetings = sum(meetings)
    while totalmeetings:
        l = totalmeetings
        for i in range(n):
            for j in range(m):
                if meetings[i] > 0 and x[i][j] < ubound[i][j]:
                    if week_load(x, j, n, h)+h[i][sum(x[i][0:j+1])]<Q[j]:
                        x[i][j] += 1
                        meetings[i] -= 1
                        totalmeetings -= 1
                        if meetings[i] == 0:
```

```
                        break
            if l == totalmeetings:
                break
        if totalmeetings:
            maximun = Q.index(max(Q))
            maxIndexes = [j for j in range(m) if Q[j] == Q[maximun]]
            for i in range(n):
                while meetings[i] <> 0:
                    l = meetings[i]
                    for j in maxIndexes:
                        if x[i][j] < ubound[i][j]:
                            x[i][j] += 1
                            meetings[i] -= 1
                            if meetings[i] == 0:
                                break
                    if meetings[i] == l:
                        raise ValueError,i
        return Solution(x, n, m)
```

Código en Python 3.2: Implementación de una solución inicial para el balance de carga docente (initial.py)

```
from pybrain.structure.evolvables.evolvable import Evolvable
class Solution(Evolvable):
    def __init__(self, x, n, m):
        if len(x) <> n*m:
            self.x = []
            for item in x:
                self.x.extend(item)
        else:
            self.x = x
        self.n = n
        self.m = m
        self.fitness = None
    def mutate(self, i, s1, s2):
        self.x[i*self.m+s1] += 1
        self.x[i*self.m+s2] -= 1
    def copy(self):
        return Solution(list(self.x), self.n, self.m)
    def __getitem__(self,i):
        return self.x[i*self.m:(i+1)*self.m]
    def __ne__(self, obj):
        return self.x <> obj
    def __eq__(self, obj):
        return not self.x <> obj.x
    def __repr__(self):
        return str(self.x)
```

Código en Python 3.3: Implementación de la clase Solution para el balance de carga docente (Solution.py)

```
# -*- coding: cp1252 -*-
from tabu import TabuSearch
from random import choice, sample
class BTabu(TabuSearch):
    N, M = 1, 1
    def __init__(self, n, m, h, p, Q, initEvaluable, lbound = None,\
                 ubound = None,**kwargs):
        self.n, self.m, self.h, self.p, self.Q = range(n), range(m), h,\
                                                 p, Q
        if not lbound:
            self.lbound = [[0]*m]*n
        else:
```

```
            self.lbound = lbound
        if not ubound:
            self.ubound = [[5]*m]*n
        else:
            self.ubound = ubound
        maximun = Q.index(max(Q))
        self.Jmax = [j for j in self.m if self.Q[j] == self.Q[maximun]]
        TabuSearch.__init__(self, evaluator = self.objective,\
                            initEvaluable = initEvaluable, **kwargs)
    def week_load(self, solution, j): # Implementación de la fórmula Cj
        load = 0
        for i in self.n:
            sup = 0
            for s in range(j+1):
                sup += solution[i][s]
            inf = 0
            for s in range(j):
                inf += solution[i][s]
            for k in range(inf, sup):
                load += self.h[i][k]
        return load
    def moves(self, solution):# Vecindad de tamaño N
        neighborhood, recent, neighborhoodSize, counter = [], [], 0, 0
        while neighborhoodSize <> self.N and counter <> self.M:
            i = choice(self.n)
            s1, s2 = sample(self.m, 2)
            if (i, s1, s2,) not in recent\
               and ((i, s1, s2,) not in self.tabuList)\
               and solution[i][s2]-1 >= self.lbound[i][s2]\
               and solution[i][s1]+1 <= self.ubound[i][s1]:
                candidate = solution.copy()
                candidate.mutate(i, s1, s2)
                week_hour_constraint = True
                for j in range(min(s1,s2)+(s2<s1 and 1 or 0),\
                               max(s1,s2)+(s2<s1 and 1 or 0)):
                    if j not in self.Jmax:
                        if self.week_load(candidate,j)>self.Q[j]:
                            week_hour_constraint = False
                            break
                if week_hour_constraint:
                    neighborhood.append((candidate, (i, s2, s1,)))
                    neighborhoodSize += 1
```

```
                recent.append((i, s1, s2,))
                counter += 1
        return neighborhood
    def hour_found(self, solution, i):
        hourFound = 0
        for j in self.m:
            sup = 0
            for s in range(j+1):
                sup += solution[i][s]
            inf = 0
            for s in range(j):
                inf += solution[i][s]
            for k in range(inf, sup):
                hourFound += self.h[i][k]
        return hourFound
    def objective(self, solution): # Función objetivo
        avrg, loads = 0, []
        for j in self.m:
            loads.append(self.week_load(solution, j))
        avrg = sum(loads)/float(len(self.m))
        obj = 0
        for load in loads:
            obj += pow(load-avrg, 2)
        return obj
```

Código en Python 3.4: Codificación de TS para el balance de carga docente (BTabu.py)

Estos algoritmos pueden ser corridos directamente desde la línea de comandos de Python y pueden ser empleados para reproducir y/o confirmar los resultados numéricos que el autor de este trabajo ha presentado. Un ejemplo de utilización es el siguiente:

Considérese que un grupo hipotético tiene en un determinado semestre 8 asignaturas a ser impartidas en dieciséis semanas. Se asume, por simplicidad en la ilustración del ejemplo, que todos los turnos de clase son constantes e iguales a dos horas. La tabla 3.6 muestra el fondo de tiempo de cada una de estas asignaturas así como la cantidad de turnos de clase a impartir durante las dieciséis semanas que conforman este semestre.

Tabla 3.6: Un ejemplo de balance de carga docente desde la línea de comandos de Python

Asignatura	Fondo de tiempo	Cantidad de turnos de clase
A	48	24
B	48	24
C	96	48
D	54	27
E	32	16
F	60	30
G	80	40
H	32	16

Los siguientes segmentos de código ilustran cómo pueden introducirse, desde la línea de comandos de Python o mediante un script de Python, los datos para correr un balance de carga docente.

Primero se importan los módulos **BTabu** e **initial** que contienen las implementaciones de TS para el balance de carga docente y la obtención de una solución inicial respectivamente:

```
from BTabu import BTabu
from initial import initial
```

A continuación se inicializan los parámetros: n, m, p, h, Q, *lbound* y *ubound*, que contienen: la cantidad de asignaturas, cantidad de semanas del semestre, cantidad de turnos de clase de cada asignatura, duración (en horas) de los turnos de clase de cada asignatura, la carga máxima permitida para cada semana del semestre, la cota inferior para la cantidad de turnos de clase de cada asignatura y la cota superior para la cantidad de turnos de clase semanal de cada asignatura respectivamente.

```
"""Cantidad de asignaturas"""
n = 8
"""Cantidad de semanas del semestre"""
m = 16
"""Especificar en la variable p la cantidad de encuentros de cada
asignatura"""
p = [24, 24, 48, 27, 16, 30, 40, 16]
"""Especificar en la variable h la cantidad de horas que tiene cada
encuentro de cada asignatura"""
h = [[2]*24, [2]*24, [2]*48, [2]*27, [2]*16, [2]*30, [2]*40, [2]*16]
"""Carga máxima permitida en cada semana"""
Q = [36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36]
"""Como mínimo se imparten cero turnos de clase de cada asignatura
en una semana"""
lbound = [[0]*m]*n
"""Como máximo se imparten seis turnos de clase de cada asignatura
en una semana"""
ubound = [[6]*m]*n
```

Para obtener una solución inicial del problema se utiliza la función **initial**, pasando como argumentos para esta función los parámetros n, m, p, Q, h, *lbound* y *ubound*. Se crea una instancia del problema mediante la función **BTabu** y se asigna a la variable *balance*. El resto de los detalles pueden seguirse en la propia documentación que aparece en el código.

```
"""Obtención de una solución inicial (posiblemente infactible)"""
sol = initial(n, m, p, Q, h, lbound, ubound)
"""Se crea una instancia del balance de carga docente"""
balance = BTabu(n = n, m = m, h = h, p = p, Q = Q,initEvaluable = sol,\
                lbound = lbound, ubound = ubound)
"""Para proporcionar información durante el proceso de optimización"""
balance.verbose = True
"""Tamaño de la vecindad de una solución"""
balance.N = 50
"""Realizar M intentos para generar una vecindad"""
balance.M = 100
"""Se desea minimizar la función objetivo del modelo"""
balance.minimize = True
"""El algoritmo se detiene si se alcanza esta evaluación"""
balance.desiredEvaluation = 0
"""Realizar un máximo de 50 iteraciones para encontrar el óptimo"""
balance.learn(50)
```

Para visualizar los resultados del balance se utiliza el segmento de código siguiente:

```
"""Muestra como quedan balanceadas las semanas"""
r = balance.bestSoFar
week_balance = "%d"+"|%d"*(m-1)
b = []
for j in range(m):
    b.append(balance.week_load(r, j))
week_balance = week_balance%tuple(b)
print "Balance semanal: ",week_balance
print "Valor de la función objetivo = ", balance.bestEvaluation
```

Los resultados obtenidos al correr este script desde la línea de comandos de Python son el siguientes:

```
Balance semanal:  30|28|28|28|28|28|28|28|28|28|28|28|28|28|28
Valor de la función objetivo =  3.75
```

Puede apreciarse que prácticamente todas las semanas del semestre tienen la misma cantidad de horas, lo que demuestra que se ha obtenido un buen balance de carga docente. El código completo de este ejemplo puede ser revisado en el anexo A.

3.6. Conclusiones del capítulo

1. La utilización del modelo heurístico y el algoritmo TS propuestos en este trabajo permite realizar balances de carga docente que poseen un mejor equilibrio de horas clases semanales respecto a aquellos confeccionados manualmente por el personal de planificación docente de la Universidad de Cienfuegos.

2. La utilización del modelo heurístico y el algoritmo TS propuestos en este trabajo reduce a cero los errores del balance de carga docente obtenido y disminuye significativamente (de dos días a aproximadamente 40 segundos) el tiempo de planificación de la carga docente de un grupo.

3. El algoritmo TS propuesto converge generalmente a un óptimo global cuando se aplica a la solución del problema de balance de carga docente con turnos de clase de duración constante.

4. Los resultados numéricos mostrados en el anexo D parecen indicar que el algoritmo TS propuesto conserva la propiedad de convergencia global cuando éste se aplica a la solución del problema de balance de carga docente con turnos de clase de duración arbitraria.

Conclusiones Generales

1. Se formula un modelo matemático heurístico general para la modelación del problema de balance de carga docente en la Universidad de Cienfuegos.

2. Se propone un algoritmo de solución basado en el algoritmo TS para la solución del modelo matemático heurístico general.

3. De la experimentación numérica se concluye que: La utilización del modelo y el algoritmo propuestos en este trabajo permite realizar balances de carga docente que poseen un mejor equilibrio de horas clases semanales respecto a aquellos confeccionados manualmente por el personal de planificación docente, reduce a cero los errores del balance de carga docente obtenido y disminuye significativamente (de dos días a aproximadamente 40 segundos) el tiempo de planificación de la carga docente de un grupo.

4. Del análisis de convergencia realizado se concluye que: El algoritmo propuesto converge generalmente a un óptimo global cuando se aplica a la solución del problema de balance de carga docente con turnos de clase de duración constante y además, existe evidencia numérica para suponer que éste conserva la propiedad de convergencia global cuando se consideran turnos de clase de duración arbitraria.

5. Para la automatización de la solución del problema de balance de carga docente se utilizó el lenguaje de programación Python, las bibliotecas numéricas PulP y PyBrain así como la biblioteca para la programación de la interfaz gráfica de usuario PyQt. Esto permitió completar en tiempo una aplicación de escritorio sencilla pero útil que asiste a los planificadores en la confección de los balances de carga docente.

Recomendaciones

1. Analizar la posibilidad de comprobar la convergencia global del algoritmo propuesto para el caso de turnos de clases de duración arbitraria tomando como solución inicial para un solucionador exacto, la solución dada por el algoritmo propuesto.

2. Integrar de manera efectiva un algoritmo de generación de horarios docentes y el algoritmo propuesto para la solución del problema de balance de carga docente en un sistema único de planificación docente.

3. Aunque ya se viene utilizando una versión anterior de la aplicación *QBalance* para la confección de balances de carga docente desde aproximadamente un año y medio, resulta importante analizar si los planificadores docentes necesitan incorporar otros requisitos que pudieran cambiar el modelo y el algoritmo propuestos.

4. Integrar el sistema de planificación docente descrito en la recomendación dos con el sistema SIGENU, para que pueda ser utilizado en otras universidades del país.

Referencias

Alinia, M., Taghi, M., y Baghmisheh, V. (2012). Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem. *Swarm and Evolutionary Computation*, 1–14. Descargado de `http://dx.doi.org/10.1016/j.swevo.2012.06.004` doi: 10.1016/j.swevo.2012.06.004

Al-milli, N. R. (2010). Hybrid Genetic Algorithms with Great Deluge For Course Timetabling. *Journal of Computer Science*, *10*(4), 283–288.

Andrei, N. (2008). An unconstrained optimization test functions collection. *Adv. Model. Optim*, *10*(1), 147–161.

Azadeh, A., Gholizadeh, H., y Jeihoonian, M. (2013). A multi–objective optimisation model for university course timetabling problem using a mixed integer dynamic non–linear programming. *International Journal of Services and Operations Management*, *15*(4), 467–481.

Bai, R. (2005). *An investigation of novel approaches for optimising retail shelf space allocation* (Tesis Doctoral no publicada). University of Nottingham.

Bazaraa, M., y Elshafei, A. (1979). An exact branch-and-bound procedure for the quadratic-assignment problem. *Naval Research Logistics Quarterly*, *26*(1), 109–121.

Blum, C., y Roli, A. (2003). Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison. *ACM Computing Surveys (CSUR)*, *35*(3), 268–308.

Burkard, R. E., y Offermann, D. M. J. (1977). Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research*, *21*(4), B121–B132.

Burke, E. K., y Kendall, G. (2005). *SEARCH METHODOLOGIES. Introductory Tutorials in Optimization and Decision Support Techniques*. New York: Springer Science.

Burke, E. K., Kendall, G., y Soubeiga, E. (2003). A Tabu-Search Hyperheuristic for Timetabling. *Journal of Heuristics*, *9*, 451–470.

Cartaya, Y. E. (2009). *Sistema Informático para la Confección de Horarios Docentes en la Facultad de Informática de la Universidad de Cienfuegos* (tesis de grado). Universidad de Cienfuegos “Carlos Rafael Rodríguez”.

Cartaya, Y. E. (2013). *Sistema de apoyo a la planificación de Horarios Docentes* (tesis de maestría). Universidad Central ”Marta Abreu”de Las Villas.

Castro, C., y Manzano, S. (2001). Variable and value ordering when solving balanced academic curriculum problems. *arXiv preprint cs/0110007*.

Cetin Demirel, N., y Duran Toksari, M. (2006). Optimization of the quadratic assignment problem using an ant colony algorithm. *Applied Mathematics and Computation*, *183*, 427–435. doi: 10.1016/j.amc.2006.05.073

Charnes, A., Cooper, W. W., DeVoe, J., Learner, D. B., y Reinecke, W. (1968). A goal programming model for media planning. *Management Science*, *14*(8), B–423.

Cook, S. A. (1971). The complexity of theorem-proving procedures. En *Proceedings of the third annual acm symposium on theory of computing* (pp. 151–158).

Dantzig, G. B. (1951). Maximization of a Linear Function of Variables Subject to Linear Inequalities, in Activity Analysis of Production and Allocation.

de desarrollo SIGENU-CUJAE, E. (2010). *Sistema de información docente de la educación superior cubana.* `http://sigenu.mes.edu.cu:8080/dmmes/pages/info/aboutUsWelcome.faces`. (Accessed: 2014-02-06)

de Klerk, E., Sotirov, R., y Truetsch, U. (2013). A new semidefinite programming relaxation for the quadratic assignment problem and its computational perspectives.

Deris, S., Zaiton, S., y Hashim, M. (2009). Incorporating Of Constraint-Based Reasoning Into Particle Swarm Optimization For University Timetabling Problem. *Science*, *1*(June), 1–21.

Dictionary.com. (2014, 19 de Feb). *The free on-line dictionary of computing.* `http://dictionary.reference.com/browse/np-hard`. (Accessed: 2014-02-19)

Eberhart, R. C., y Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the sixth international symposium on micro machine and human science*, 39–43.

Elshafei, A.Ñ. (1977). Hospital layout as a quadratic assignment problem*. *Journal of the Operational Research Society*, *28*(1), 167–179.

Gallardo, A., y Lowther, D. A. (2000). Some Aspects of Niching Genetic Algorithms Applied to Electromagnetic Device Optimization. *IEEE TRANSACTIONS ON MAGNETICS*, *36*(4), 1076–1079.

Gendreau, M. (2003). An introduction to tabu search. En F. Glover y G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 37–54). Kluwer Academic Publishers.

Geoffrion, A. M., y Graves, G. (1976). Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/lp approach. *Operations Research*, *24*(4), 595–610.

Glover, F., y Kochenberger, G. A. (2003). *Handbook of Metaheuristics (International series in operations research & management science)*. Springer Berlin.

Glover, F., y McMillant, C. (1986). THE GENERAL EMPLOYEE SCHEDULING PROBLEM : AN INTREGRATION OF MS AND AI. *Computers & Operations Research*, *13*(5), 563–573.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, *64*(5), 275–278.

Gomory, R. E. (1960). Solving linear programming problems in integers. *Combinatorial Analysis*, *10*, 211–215.

Holland, J. H. (1975). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press.

Jones, K. O. (2006). COMPARISON OF GENETIC ALGORITHMS AND PARTICLE SWARM OPTIMISATION FOR FERMENTATION FEED PROFILE DETERMINATION. *International Conference on Computer Systems and Technologies- CompSysTech'2006*.

Koopmans, T. C., y Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica: journal of the Econometric Society*, 53–76.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83–97.

Lambert, T., Castro, C., Monfroy, E., y Saubion, F. (2006). Solving the balanced academic curriculum problem with an hybridization of genetic algorithm and constraint propagation. En *Artificial intelligence and soft computing–icaisc 2006* (pp. 410–419). Springer.

Land, A. H., y Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, *28*(3), 497–520.

Lao, J. M. I., y Gómez, M. E. P. (2007). Las tecnologías de la información y las comunicaciones (TIC) en la gestión académica del proceso docente educativo en la educación superior. *Pedagogía Universitaria*, *12*(1), 58–68.

Leong, W.-f., y Yen, G. G. (2008). PSO-Based Multiobjective Optimization Adaptive Local Archives. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on System, Man and Cybernetics*, *38*(5), 1270–1293. doi: 10.1109/TSMCB.2008.925757

Michael, R. G., y Johnson, D. S. (1979). Computers and intractability: A guide to the

theory of np-completeness. *WH Freeman & Co., San Francisco*.

Michalewicz, Z., y Fogel, D. B. (2000). *How to solve it: Modern Heuristics* (2.ª ed.). New York: Springer.

Mitchell, S., O'Sullivan, M., y Dunning, I. (2011). Pulp: A linear programming toolkit for python.

Nagar, A., Heragu, S. S., y Haddock, J. (1996). A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem. *Annals of Operations Research*, *63*(3), 397–414.

Naseem, S., y Shengxiang, J. (2010). A Hybrid Genetic Algorithm and Tabu Search Approach for Post Enrolment Course Timetabling. *Journal of Scheduling*, 1–19.

Padberg, M., y Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, *33*(1), 60–100.

Panigrahi, B. K., Shi, Y., y Lim, M.-h. (2011). *Handbook of Swarm Intelligence* (M.-H. Lim y Y.-S. Ong, Eds.). Springer-Verlag Berlin Heidelberg. doi: 10.1007/978-3-642-17390-5

Paquet, U., y Engelbrecht, A. P. (2003). A new particle swarm optimiser for linearly constrained optimisation. En *Evolutionary computation, 2003. cec'03. the 2003 congress on* (Vol. 1, pp. 227–233).

Perzina, R., y Ramik, J. (2013). Timetabling problem with fuzzy constraints: A self-learning genetic algorithm. *constraints*, *3*(4).

Ramkumar, A. S., y Ponnambalam, S. G. (2006). Hybrid ant colony system for solving quadratic assignment formulation of machine layout problems. En *Cybernetics and intelligent systems, 2006 ieee conference on* (pp. 1–5).

Riverbankcomputing. (2014). *What is PyQt?* `http://www.riverbankcomputing.com/software/pyqt/intro`. (Accessed: 2014-03-13)

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *The Computer Journal*, *3*(3), 175–184.

Saaty, T. L. (1980). The analytic hierarchy process: planning, priority setting, resources allocation. *M cGraw-Hill*.

Sahni, S., y Gonzalez, T. (1976). P-complete approximation problems. *Journal of the ACM (JACM)*, *23*(3), 555–565.

Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., ... Schmidhuber, J. (2010). Pybrain. *The Journal of Machine Learning Research*, *11*, 743–746.

Student, B. (1908). The probable error of a mean. *Biometrika*, *6*(1), 1–25.

Taillard, E. (1991). Robust taboo search for the quadratic assignment problem. *Parallel computing*, *17*(4), 443–455.

Tamayo, S. C., Campaña, C. M. P., y Expósito, C. F. R. (2007). Alternativa para el proceso de planificación de horarios docentes de una universidad. *Ciencias Holguín*, *13*(4).

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P., y Dizbay, I. E. (2013). Metaheuristic algorithms for the quadratic assignment problem. En *Computational intelligence in production and logistics systems (cipls), 2013 ieee workshop on* (pp. 131–137).

Tseng, L.-y. (2006). A Hybrid Metaheuristic for the Quadratic Assignment Problem. *Computational Optimization and Applications*, *34*, 85–113. doi: 10.1007/s10589-005-3069-9

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics*, *1*(6), 80–83.

Xia, W.-j., y Wu, Z.-m. (2006). A hybrid particle swarm optimization approach for the job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, *29*(3-4), 360–366.

Yin, P.-y. (2004). A discrete particle swarm algorithm for optimal polygonal approximation of digital curves. *Journal of Visual Communication & Image Representation*, *15*, 241–260. doi: 10.1016/j.jvcir.2003.12.001

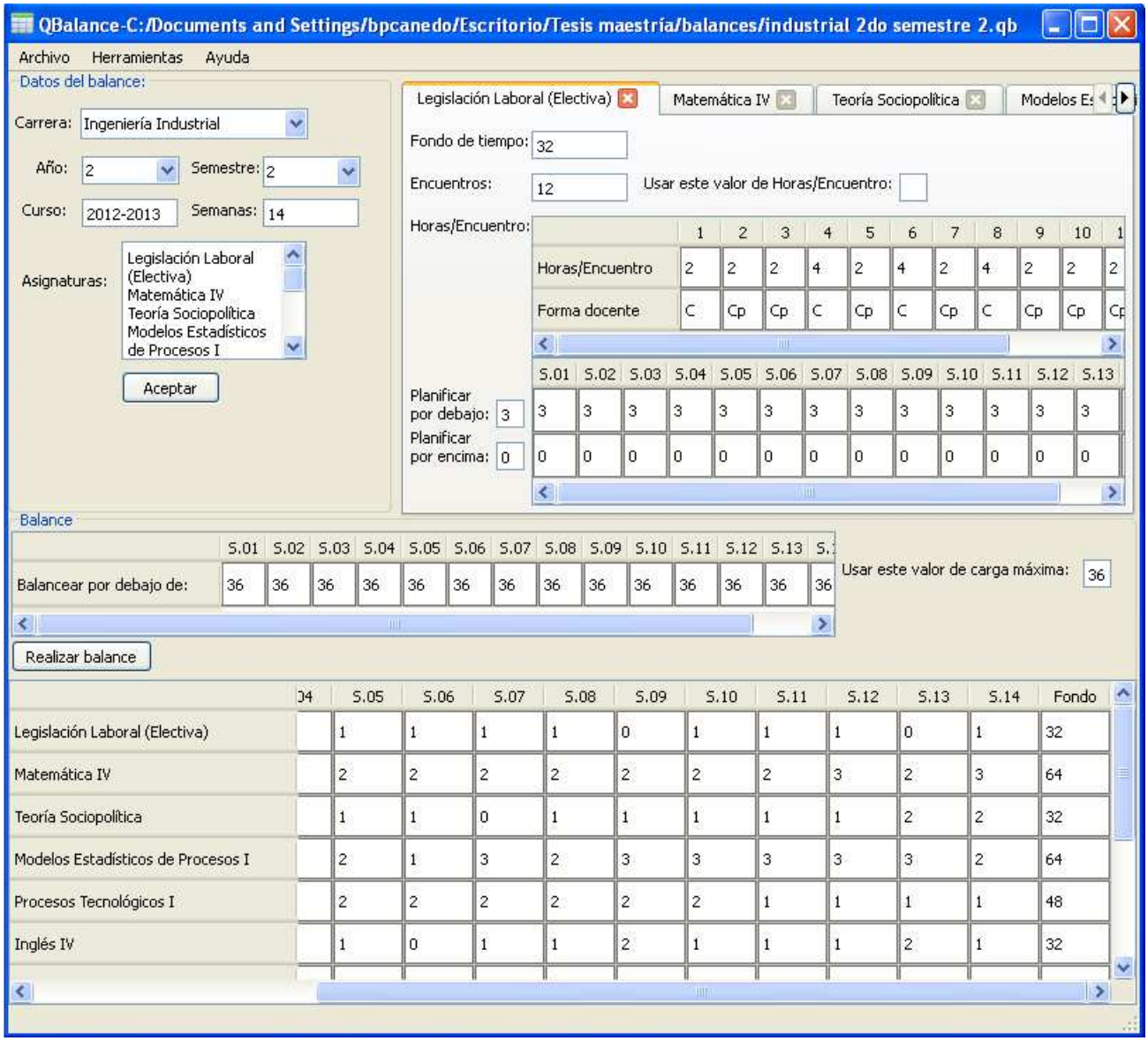
Ejemplo de solución desde la línea de comandos de Python

```
# -*- coding: cp1252 -*-
from BTabu import BTabu
from initial import initial
n = 8
m = 16
p = [24, 24, 48, 27, 16, 30, 40, 16]
h = [[2]*24, [2]*24, [2]*48, [2]*27, [2]*16, [2]*30, [2]*40, [2]*16]
Q = [36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36]
lbound = [[0]*m]*n
ubound = [[6]*m]*n
sol = initial(n, m, p, Q, h, lbound, ubound)
balance = BTabu(n = n, m = m, h = h, p = p, Q = Q,initEvaluable = sol,\
                lbound = lbound, ubound = ubound)
balance.verbose = True
balance.N = 50
balance.M = 100
balance.minimize = True
balance.desiredEvaluation = 0
balance.learn(50)
r = balance.bestSoFar
week_balance = "%d"+"|%d"*(m-1)
b = []
for j in range(m):
    b.append(balance.week_load(r, j))
week_balance = week_balance%tuple(b)
print "Balance semanal: ",week_balance
print "Valor de la función objetivo = ", balance.bestEvaluation
```

Código en Python A.1: Ejemplo de solución desde la línea de comandos de Python

Anexo B

QBalance, una aplicación de escritorio para la solución del problema de balance de carga docente



Anexo C

Ejemplo de salida de QBalance (Open Document Format)

Institución: Universidad "Carlos Rafael Rodríguez" de Cienfuegos
Departamento: Planificación docente
Documento: Balance de carga docente
Curso: 2012-2013. **Semestre:** 2. **Carrera:** Ingeniería Industrial. **Año académico:** 2.

	Legislación Laboral (Electiva)	Matemática IV	Teoría Sociopolítica	Modelos Estadísticos de Procesos I	Procesos Tecnológicos I	Inglés IV	Física III	Defensa Nacional	Debate	Base de Datos	Carga
1	C1 (2h)	C1 (2h) Cp1 (2h) Cp2 (2h)	C1 (2h)	C1 (2h) Cp1 (2h) Cp2 (2h)	C1 (2h) Cp1 (2h)	C1 (2h)		C1 (2h) C2 (2h)	C1 (2h)	C1 (2h) Cp1 (2h)	32
2	Cp1 (2h)	L1 (2h) C2 (2h) Cp3 (2h)	Cp1 (2h)	L1 (2h) E1 (2h) C2 (2h)	Cp2 (2h) C2 (2h) Cp3 (2h)	Cp1 (2h)		C3 (2h) C4 (2h)		Cp2 (2h) L1 (2h)	32
3	Cp2 (2h)	Cp4 (2h) Cp5 (2h) L2 (2h)	Cp2 (2h) C2 (2h)	Cp3 (2h) Cp4 (2h) L2 (2h)	Cp4 (2h)	Cp2 (2h)	C1 (3h)	C5 (2h) C6 (2h)	C2 (2h)	C2 (2h)	33
4	C2 (4h)	E1 (2h) C3 (2h) Cp6 (2h)	Cp3 (2h)	C3 (2h) Cp5 (2h)	C3 (2h)	L1 (2h)	Cp1 (2h) Cp2 (2h)	C7 (2h) C8 (2h)		Cp3 (2h) Cp4 (2h)	32
5	Cp3 (2h)	C4 (2h) Cp7 (2h) Cp8 (2h)	Cp4 (2h)	Cp6 (2h) L3 (2h) E2 (2h)	Cp5 (2h) Cp6 (2h)	E1 (2h)	C2 (2h) Cp3 (2h)	C9 (2h)	C3 (2h)	L2 (2h)	32
6	C3 (4h)	L3 (2h) E2 (2h)	C3 (2h)	C4 (2h) Cp7 (2h) Cp8 (2h)	Cp7 (2h) E1 (2h)	C2 (2h)	Cp4 (3h)	C10 (2h) C11 (2h)		E1 (2h) C3 (2h)	33
7	Cp4 (2h)	C5 (2h) Cp9 (2h)	Cp5 (2h)	L4 (2h) C5 (2h) Cp9 (2h)	C4 (2h)	Cp3 (2h)	Cp5 (2h) L1 (2h)	C12 (2h) C13 (2h)	C4 (2h)	Cp5 (2h) Cp6 (2h)	32
8	C4 (4h)	Cp10 (2h) Cp11 (2h) L4 (2h) C6 (2h)		Cp10 (2h) L5 (2h) E3 (2h)	Cp8 (2h) C5 (2h)		L2 (2h) E1 (3h) C3 (3h)		C5 (2h)		32
9	Cp5 (2h)	Cp12 (2h) Cp13 (2h)	Cp6 (2h)	C6 (2h) Cp11 (2h)	Cp9 (2h) Cp10 (2h)	Cp4 (2h)	Cp6 (2h) Cp7 (2h)	C14 (2h) C15 (2h)	C6 (2h)	L3 (2h) E2 (2h)	32
10	Cp6 (2h)	Cp14 (2h) L5 (2h)	E1 (4h)	Cp12 (2h) Cp13 (2h)	Cp11 (2h) E2 (2h)	Cp5 (2h)	C4 (2h) Cp8 (2h)	C16 (2h) C17 (2h)		C4 (2h) Cp7 (2h)	32
11	Cp7 (2h)	E3 (2h) C7 (2h)	C4 (2h)	L6 (2h) C7 (2h)	C6 (2h) Cp12 (2h)	L2 (2h)	Cp9 (2h) C5 (2h) Cp10 (2h)	C18 (2h)	C7 (2h)	Cp8 (2h) C5 (2h)	32
12	E1 (4h)	Cp15 (2h) Cp16 (2h) L6 (2h)	Cp7 (2h)	Cp14 (2h) Cp15 (2h) L7 (2h)			Cp11 (2h) L3 (2h) L4 (2h) E2 (2h)			Cp9 (2h) Cp10 (2h) Cp11 (2h)	32
13					C7 (2h) Cp13 (2h) L1 (2h) E3 (2h)	E2 (2h) C3 (2h) Cp6 (2h)	C6 (2h) Cp12 (2h) Cp13 (2h) Cp14 (2h)	C19 (2h) C20 (2h) C21 (2h) C22 (2h)	C8 (2h)		32
14			C5 (2h) Cp8 (2h) Cp9 (2h)			Cp7 (2h) L3 (2h) E3 (2h)	C7 (2h) Cp15 (2h) L5 (2h) E3 (2h)			L4 (2h) L5 (2h) E3 (2h)	26
Fondo	32	64	32	64	48	32	64	44	16	48	444/444

Anexo D

Comportamiento del algoritmo TS propuesto en problemas con turnos de clase de duración arbitraria

Semanas	Instancia 1	Instancia 2	Instancia 3	Instancia 4	Instancia 5	Instancia 6
1	26	32	26	29	19	29
2	24	32	26	29	20	29
3	26	33	26	28	19	30
4	26	32	28	30	18	30
5	26	32	26	28	19	30
6	26	33	26	29	19	29
7	26	32	26	28	19	31
8	26	32	26	30	19	30
9	26	32	26	29	19	30
10	26	32	27	30	19	31
11	26	32	26	28	18	29
12	26	32	26	28	19	29
13	26	32	26	28	19	31
14		26		28	18	30
15				28	18	
16				29	19	
Función Objetivo	3,69230769	36,8571429	4,30769231	9,4375	4,4375	7,71428571