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

SISTEMA DE ASIGNACIÓN DE CURSOS PARA LA ESCUELA DE INGENIERÍA INFORMÁTICA DE LA P.U.C.V

FABIÁN CLAUDIO DROGUETT SAAVEDRA

FÉLIX MIGUEL FUENTES GONZÁLEZ

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

DICIEMBRE 2009

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

SISTEMA DE ASIGNACIÓN DE CURSOS PARA LA ESCUELA DE INGENIERÍA INFORMÁTICA DE LA P.U.C.V

FABIÁN CLAUDIO DROGUETT SAAVEDRA

FÉLIX MIGUEL FUENTES GONZÁLEZ

Profesor Guía: **Guillermo Cabrera Guerrero**

Profesor Co-referente: **Iván Mercado Bermúdez**

Carrera: **Ingeniería de Ejecución en Informática**

DICIEMBRE 2009

Dedicatoria

"Queremos agradecer a nuestras familias en especial a nuestros padres, por su apoyo durante esta etapa de aprendizaje y crecimiento, a los profesores que nos guiaron y entregaron la formación académica necesaria, a nuestros compañeros y amigos que nos acompañaron durante este proceso, y en general a todos los que nos aportaron algo durante este largo periplo para llegar a ser íntegros profesionales"

Resumen

El problema de asignación de cursos conocido también como University timetabling, es un problema típico de optimización combinatorial que se enmarca en área de la investigación de operaciones, o también puede ser visto como un problema de competencia ingenieril por lo complejo que resulta de resolver.

El enfoque propuesto presenta una revisión de varios métodos conocidos de metaheurísticas aplicados al University timetabling, en el marco de los requerimientos particulares de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso (P.U.C.V), de los cuales la búsqueda tabú es seleccionada como la técnica más apropiada dadas las condiciones del problema aquí presentado. A partir de esta elección se propone un algoritmo y su respectiva implementación en la forma de un sistema de información interoperable mediante una interfaz gráfica.

PALABRAS CLAVES: Metaheurísticas, Timetabling, Búsqueda Tabú.

Abstract

The university course timetabling problem is a typical combinatorial optimization problem, which is part of the operations research field, and also can be seen as an engineering competence problem because of the complexity involved in solving it.

In this document the problem will be approached from its origins to the different types of heuristics that exist to solve it, within the requirements of the Department of Computer Engineering of the Pontificia Unibersidad Católica de Valparaíso (P.U.C.V.) With respect to the before mentioned, a small review will be issued to address this techniques. Along with reviewing in a general way some of the existing heuristics to approach this problem, a detailed review will be presented on the technique which is thought to be one of the more appropriated to solve this particular problem, the taboo search

KEY WORDS: Metaheuristics, Timetabling, Taboo Search.

Índice

Índice de Figuras

Lista de Tablas

CAPÍTULO 1:
Visión General del Proyecto

1.1 Introducción

La Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso con sus carreras de ingeniería civil informática e ingeniería de ejecución informática, se enfrenta por lo menos dos veces al año al problema de asignar recursos para los cursos que se dictan en sus dependencias, ejemplos de los anterior son las 4 salas que se encuentran en el segundo piso de del edificio IBC o el staff de profesores de planta que dictan clases en la Escuela de Ingeniería Informática. Este problema es conocido como Timetabling que se describe la asignación de recursos que son escasos dentro de un espacio limitado de periodos de tiempo como son las distintas claves horarias en que se realizan las clases. Este es un problema complejo debido a la gran cantidad de restricciones que presenta.

Los recursos deben ser planificados y estudiados de la mejor forma posible para que no se produzca alguna escasez de ellos y esto ocasione algún deterioro en la calidad de la formación que las entidades educacionales ofrecen, otro problema que se podría presentar con la mala planificación de los recursos es que se produzca una sobre oferta de ellos por lo que se verían afectadas las finanzas de las instituciones al asignar recursos de forma desmesurada.

En ésta investigación se propondrá una solución al problema práctico de la asignación de horarios de clases (Timetabling) en la universidad, presentado como un problema de carácter académico desarrollado y resuelto en el área de la ingeniería y de la investigación de operaciones.

El Timetabling universitario radica principalmente en programar un cierto número de eventos (clases) en un espacio corto de tiempo (generalmente una semana) de las asignaturas que se imparten en un periodo académico (generalmente un semestre) considerando los profesores necesarios para realizar las clases, las aulas necesarias para impartir las cátedras, y los alumnos que participan en cada una de estos eventos, además de los periodos disponibles para los eventos (claves horarias), con el fin último de hacer el mejor uso posible de los recursos disponibles.

La resolución del Timetabling entregará una herramienta mediante la cual se podrá asignar de una buena manera los recursos disponibles en la Escuela de Ingeniería Informática produciendo un beneficio a sus estudiantes y profesores ya que podrán optar a un horario cómodo para sus clases.

1.2 Definición de Objetivos

1.2.1 Objetivo General

- Resolver el problema de asignación de cursos mediante un sistema que implemente una metaheurística permitiendo agilizar y mejorar el proceso existente en la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso y proporcione una solución de buena calidad en un tiempo aceptable volviendo más eficiente el proceso.

1.2.2 Objetivos Específicos

- Entender las técnicas heurísticas con las cuales se puede resolver el problema.

- Resolver el problema integrando una metaheurística que proporcione una solución de alta calidad en un tiempo aceptable de ejecución.

- Proporcionar una herramienta operativa para su uso.

- Proporcionar al encargado de la asignación de los horarios una herramienta Práctica que le permita un trabajo más eficiente permitiéndole un mejor uso del tiempo en el desarrollo de sus funciones.

CAPÍTULO 2:
Estado del Arte

Para desarrollar un algoritmo basado en técnicas metaheurísticas, con el fin de encontrar una solución al problema de asignación de horarios, se vuelve fundamental conocer las técnicas mediante las cuales se puede resolver el problema.

Con este fin, en la siguiente sección, entregaremos la definición del problema además de revisar las versiones del problema y las distintas técnicas que se han propuesto para resolverlo.

2.1 Complejidad del Problema

Los distintos problemas que se tratan no tienen la misma dificultad en lo que se refiere a su resolución. De manera que dado un problema cualquiera ¿De qué forma se define si el problema es fácil o difícil de resolver? O ¿Qué significa que el problema sea fácil o difícil de resolver? De este tema trata la complejidad algorítmica, la cual establece una clasificación de los distintos tipos de problemas por su grado de dificultad de acuerdo a la complejidad computacional del algoritmo más sencillo que permite asegurar la resolución del problema [1].

Para nuestros objetivos será de gran ayuda definir lo que se entiende por un algoritmo eficiente ¿significa que un algoritmo eficiente es el que requiere un tiempo aproximado de $O(n \log n)$? No necesariamente, porque todo depende del problema a resolver. Por ejemplo un algoritmo de ordenación que requiera un tiempo de $O(n^2)$ es eficiente, sin embargo un algoritmo de multiplicación matricial que requiera un tiempo de $O(n^2 \log n)$ sería un descubrimiento asombroso. Por lo tanto se puede decir que un algoritmo es eficiente si se trata del mejor algoritmo posible para resolver un problema determinado, sin embargo esta definición sería imprecisa y resultaría difícil trabajar con ella, en algunos casos "el mejor algoritmo posible" ni siquiera existiría. Además existen problemas para los cuales el mejor algoritmo posible requeriría un tiempo demasiado grande para calcular su resultado final, incluso para problemas pequeños [2]. Los problemas se podrían dividir en tratables e intratables, los intratables serían aquellos en que se ha demostrado que no existe un algoritmo que permita resolverlo en todos los casos o existe un algoritmo pero requieren una cantidad desmesurada de tiempo computacional para resolverlo [1].

La complejidad computacional se puede dividir en clases de complejidad:

Un problema tratable, un problema de clase P, es un problema que se puede resolver siempre utilizando un número de pasos que es función polinomial del tamaño de la entrada del problema.

En resumen se puede decir que los problemas de clase P se pueden resolver en un tiempo polinomial y los intratables no se pueden resolver en tiempo polinomial [1].

Los problemas de la clase NP son los que se pueden resolver utilizando una máquina de Turing no determinista en un número de pasos polinomial. Muchos de los problemas de la clase NP son problemas muy comunes, que aparecen en forma habitual en el área de la ingeniería como lo son partición de conjuntos, diseño de redes, planificación, optimización etc. [1].

De la clase NP se pueden distinguir los problemas NP-Completos, que son unos de los más difíciles de resolver, la propiedad de los problemas NP-Completos es que todo problema de la clase NP se puede trasformar polinomialmente en él.

En problemas como el que se planteará, donde el conjunto de cálculos crece exponencialmente con el número de soluciones resulta evidente que la comprobación no se puede llevar a cabo en tiempo polinomial. Estos problemas de optimización se encuentran en clase que se denomina de ***dificultad NP***.

Para los problemas de dificultad NP no existe ningún algoritmo en tiempo polinomial que permita determinar la solución óptima al problema. Para ello se utilizan métodos aproximados mediante Heurísticas que permiten acercarse a una solución óptima, generando soluciones al problema que resulten de utilidad práctica [1].

2.2 Problema de Asignación de Horarios

El Timatabling se define como "la asignación, sujeta a restricciones, de los recursos otorgados con el propósito de ser establecidos, en un espacio de tiempo, de tal manera que satisfaga lo más cercanamente posible el conjunto de objetivos deseados "[2].

Una visión del Timetabling es aquella que, asigna un conjunto de eventos (clases, exámenes), a un número limitado de periodos de tiempo generalmente a una cierta cantidad de periodos diarios en un horizonte de tiempo de una semana sujetos a un conjunto de restricciones. Estas restricciones generalmente se dividen en dos grupos. Obligatorias o duras (tener que asistir a dos clases en el mismo horario, cada clase debe tener asignada una sala, etc.) que deben ser satisfechas a cabalidad y restricciones deseables o blandas (Asistir a una clase en el ultimo horario del día, tener muchas clases seguidas en el mismo día, etc.) que pueden o no ser satisfechas y generalmente son utilizadas para entregar un costo a la función objetivo la que va a ser optimizada.

$$\text{MIN } Z = \sum_{k=1}^{p} \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ijk} X_{ijk} \tag{2.2.1}$$

Donde C_{ijk} corresponde al costo de asignar el curso k en el horario i en la sala j y donde X_{ijk} es una variable binaria que representa la existencia de la asignación, puede tomar los valores {0,1}.

De acuerdo a lo anterior se hace deseable poder contar con un sistema capaz de encontrar de forma automática una programación de horarios que se adapte de la mejor manera posible a las restricciones a partir de una descripción inicial de los salones, profesores, alumnos, y sesiones requeridas para entregar una educación de calidad.

El problema Timetabling tiene distintas versiones ya que no hay una definición estándar del problema debido a que cada institución posee una descripción particular de éste dependiendo de sus necesidades. Se han propuesto muchas versiones del problema dependiendo de la institución involucrada (universidad o colegio) y las restricciones inherentes a cada una de ellas. A continuación se dará la definición de los tipos de Timetabling propuestas [3].

Generación de horarios escolar (Class-Teacher Problem): Es la calendarización semanal de todas las clases de un colegio (básica o media), evitando que los profesores se comprometan en dos clases al mismo tiempo y viceversa. También se le conoce como modelo clase/maestro, en el cual existe una versión simplificada del problema, la cual indica que si no existe ninguna restricción el problema puede resolverse en tiempo polinomial. En la generación de horarios escolares se debe tomar en cuenta la posibilidad de que un profesor o un grupo no se encuentren disponibles en un periodo de tiempo determinado, y con esta característica el problema se convierte en NP-Completo.

Generación de horarios por curso (University Timetabling:): Se trata de la calendarización semanal de todas las clases de un conjunto de asignaturas de una universidad, minimizando la superposición de asignaturas que contienen estudiantes en común. La principal diferencia con el problema del apartado anterior consiste en que esta categoría pueden existir estudiantes en común. Si dos cursos tienen estudiantes en común entonces se dice que están en conflicto, y no pueden ser calendarizados en el mismo periodo de tiempo.

Generación de horarios por examen (Examination Timetabling): Consiste en la calendarización de los exámenes de un conjunto de asignaturas de una universidad, y esparciendo los exámenes para los estudiantes lo más posible.

El desafío en cada uno de los problemas mencionados anteriormente es desarrollar una planificación que cumpla con todas las restricciones impuestas por cada uno de los problemas individualmente y entregue una solución aceptable a sus requerimientos, transformándose en una situación de carácter compleja, considerando que las tres versiones del problema presentan una cantidad de restricciones y cálculos considerables, haciendo

que se invierta un tiempo considerable en la búsqueda de una solución que ataque real y eficientemente al problema.

2.3 Formas de Construir Una solución al Problema

Realizar el proceso de manera manual sin la aplicación de una tecnología consiste en que una o más personas deben decidir que cátedra a un profesor en un espacio de tiempo exento de toda intersección, es decir, establecer un espacio de tiempo absolutamente propio, en el cual no se interponga ninguna persona o elemento que se sobreponga a ese quehacer. Es indudable que no se puede descartar la intervención de elementos exógenos que puedan perturbar el quehacer como lo que es una planificación que altere la cotidianeidad sin dejar de mencionar que él o los procesos varían de una institución a otra o como añadidura, la solución obtenida pueden no ser satisfactoria con respecto a alguna restricción.

- En el caso de la generación de horarios interactiva, se define como la existencia de una aplicación con la cual una computadora realice el proceso, en combinación con la interacción del usuario. En este sentido el usuario puede guiar la búsqueda hacia direcciones promisorias que el sistema por sí mismo no pueda ser capaz de deducir [3].

- El proceso de la generación de horarios por lotes lo realiza exclusivamente una computadora, funcionando como una caja negra en la cual un sistema automatizado explora el espacio de búsqueda para encontrar una solución satisfactoria [3].

Con todo lo expuesto anteriormente nos podemos dar cuenta que la resolución del Timetabling no es algo trivial y más bien es un problema complejo que debe ser tratado en el área de la ingeniería para encontrarle una solución que se ajuste a las necesidades de calidad y tiempo necesarias para que una institución educativa pueda entregar un producto (educación) que se adapte a los estándares establecidos.

Ya que hemos expuesto el tipo de problema que estamos tratando y sus distintas variantes a continuación daremos una revisión a los distintos métodos (heurísticas) que se han utilizado para dar solución al problema en cuestión. Pero antes de revisar las técnicas propuestas se hará una observación a lo que son las heurísticas las que se pueden observar en mayor profundidad en [4] metaheurísticas, una visión global.

La idea más genérica del término heurística está relacionada con la tarea de resolver inteligentemente problemas reales usando el conocimiento disponible. El término heurística proviene de una palabra griega con un significado relacionado con el concepto de encontrar, y se vincula a la supuesta exclamación eureka de Arquímedes al descubrir su famoso principio.

La concepción más común en Inteligencia Artificial (**IA**) es interpretar que heurístico es el calificativo apropiado para los procedimientos que, empleando

conocimiento acerca de un problema y de las técnicas aplicables, tratan de aportar soluciones (o acercarse a ellas) usando una cantidad de recursos (generalmente tiempo) razonables. En un problema de optimización, aparte de las condiciones que deben cumplir las soluciones del problema, se busca la que es óptima según algún criterio de comparación entre ellas. En investigación operativa, el término heurístico se aplica a un procedimiento de resolución de problemas de optimización con una concepción diferente. Se califica de heurístico a un procedimiento para el que se tiene un alto grado de confianza en que encuentra soluciones de alta calidad con un coste computacional razonable, aunque no se garantice su optimalidad o su factibilidad, e incluso, en algunos casos, no se llegue a establecer lo cerca que se está de dicha situación. Se usa el calificativo heurístico en contraposición a exacto, que se aplica los procedimientos a los que se les exige que la solución aportada sea óptima o factible. Una solución heurística de un problema es la proporcionada por un método heurístico, es decir, aquella solución sobre la que se tiene cierta confianza de que es factible y óptima, o de que alcanza un alto grado de optimalidad y/o factibilidad. También es usual aplicar el término heurística cuando, utilizando el conocimiento que se tiene del problema, se realizan modificaciones en el procedimiento de solución del problema que, aunque no afectan a la complejidad del mismo, mejoran el rendimiento en su comportamiento práctico.

Algunas heurísticas para resolver un problema de optimización pueden ser más generales o específicas que otras. Los métodos heurísticos específicos deben ser diseñados a propósito para cada problema, utilizando toda la información disponible y el análisis teórico del modelo. Los procedimientos específicos bien diseñados suelen tener un rendimiento significativamente más alto que las heurísticas generales. Las heurísticas más generales, por el contrario, presentan otro tipo de ventajas, como la sencillez, adaptabilidad y robustez de los procedimientos. Sin embargo, las heurísticas generales emanadas de las metaheurísticas pueden mejorar su rendimiento utilizando recursos computacionales y estrategias inteligentes.

El término metaheurísticas se obtiene de anteponer a heurística el sufijo meta que significa “más allá” o a “un nivel superior”. Los conceptos actuales de lo que es una metaheurística están basados en las diferentes interpretaciones de lo que es una forma inteligente de resolver un problema. Las metaheurísticas son estrategias inteligentes para diseñar o mejorar procedimientos heurísticos muy generales con un alto rendimiento. El término metaheurística apareció por primera vez en el artículo seminal sobre Búsqueda Tabú de Fred Glover en 1986. A partir de entonces han surgido multitud de propuestas de pautas para diseñar buenos procedimientos para resolver ciertos problemas que, al ampliar su campo de aplicación, han adoptado la denominación de metaheurísticas.

2.4 Técnicas de Resolución Utilizadas

En la siguiente sección se hará una revisión de las principales características de los métodos mediante los cuales se ha resuelto el Timetabling universitario profundizando en el que metaheurística se adapta de mejor manera al problema que se ha expuesto y se pretende resolver.

2.4.1 Enfriado Simulado (Simulated Annealing)

Este tipo de algoritmos aparece a principios de los años 80. Se trata de un tipo de modelo de resolución para la optimización de problemas de tipo combinatorio con mínimos locales. Su aproximación consiste en generar aleatoriamente una solución cercana a la solución actual (o en el entorno de la solución) y la acepta como buena si consigue reducir una determinada función de coste, o con una determinada probabilidad de aceptación. Esta probabilidad de aceptación se irá reduciendo con el número de iteraciones y está relacionada también con el grado de empeoramiento del coste.

Este tipo de algoritmos se diferencia de los algoritmos de Hillclimbing en que en estos últimos solo se acepta una nueva solución si ésta mejora la anterior de manera que cada solución que se obtenga vaya obteniendo soluciones mejores cada vez. El problema de este tipo de algoritmos es que la solución final suele encontrarse en un mínimo local no explorando con suficiente amplitud el espacio de soluciones. Sin embargo, en la aplicación del algoritmo de enfriamiento lento se pueden aceptar soluciones que empeoran la solución actual, sólo que esta aceptación dependerá de una determinada probabilidad que depende de un parámetro, denominado temperatura, del estado del sistema.

Estos algoritmos se llaman de esta forma por su parecido en funcionamiento al proceso de enfriamiento metalúrgico: cuando se enfría un metal fundido suficientemente despacio, tiende a solidificarse en una estructura de mínima energía (equilibrio térmico); según va disminuyendo la temperatura, las moléculas del metal tienen menor probabilidad de moverse de su nivel energético [1].
Los algoritmos de enfriamiento lento simulado tienen algunas ventajas con respecto a otras técnicas de optimización global. Entre las ventajas de estos algoritmos se pueden citar:

- Lo relativamente sencillo que resulta implementar este tipo de problemas.
- Su aplicabilidad a la mayoría de los problemas de optimización con una estructura combinatoria.
- Su capacidad para ofrecer soluciones razonablemente buenas a la mayoría de los problemas, aunque hay una cierta dependencia de la planificación del enfriamiento y los movimientos que se realicen.
- La facilidad con la que se puede combinar este tipo de algoritmos con otras técnicas heurísticas como los sistemas expertos, los algoritmos genéticos, las redes neuronales, etc., consiguiendo sistemas híbridos que pueden resultar de gran potencia en la resolución de problemas muy complejos.

Por otra parte, también se pueden citar algunos aspectos que pueden limitar su utilización:

- Se necesita elegir con mucho cuidado los movimientos que se realizan, así como los parámetros que se van a utilizar para tratarlo, como por ejemplo la tasa de enfriamiento.

- Una ejecución del problema puede requerir mucho tiempo de cálculo.

- Puede que sea necesario realizar muchas ejecuciones para encontrar una solución satisfactoria.

- Dependiendo de los parámetros elegidos, las soluciones que se van encontrando pueden ser poco estables, "saltando" mucho de unas a otras sin encontrar una solución buena con la rapidez suficiente lo que obliga a retocar los parámetros con las distintas ejecuciones.

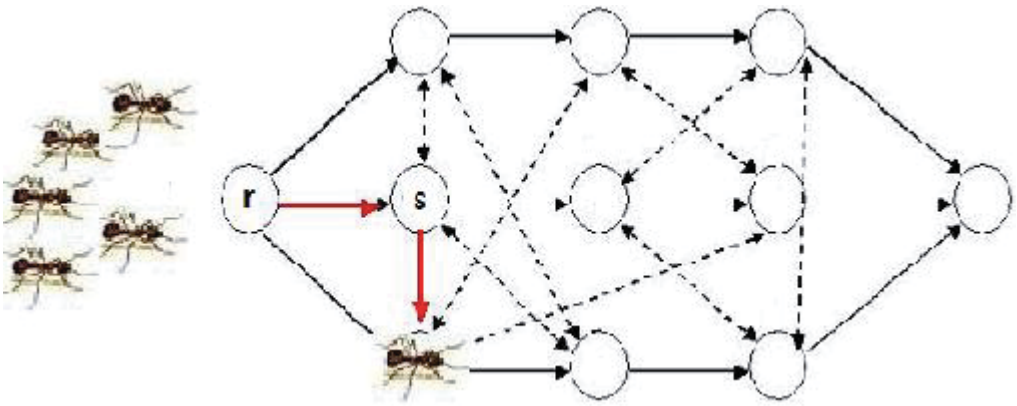
2.4.2 Algoritmos de Hormigas

Los algoritmos de ACO (Ant Colony Optimization) se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria. Se basan esencialmente en una colonia de hormigas artificiales, estos son, unos agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales [5].

Los algoritmos de ACO son esencialmente algoritmos constructivos, en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- Información heurística, que mide la preferencia heurística de moverse desde el nodo r hasta el nodo s, o sea, de recorrer la arista A_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.

- Información de los rastros de feromona artificiales, que mide la "deseabilidad aprendida" del movimiento de r a s. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas.



El modo de operación básico de un algoritmo de ACO es como sigue: las m hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema (que puede representarse en forma de grafo con pesos). Este movimiento se realiza siguiendo una regla de transición que está basada en la información local disponible en las componentes (nodos). Esta información local incluye la información heurística y memorística (rastros de feromona) para guiar la búsqueda. Al moverse por el grafo de construcción, las hormigas construyen incrementalmente soluciones. Opcionalmente, las hormigas pueden depositar feromona cada vez que crucen un arco (conexión) mientras que construyen la solución (actualización en línea paso a paso de los rastros de feromona). Una vez que cada hormiga ha generado una solución se evalúa ésta y puede depositar una cantidad de feromona que es función de la calidad de su solución [5] (actualización en línea a de los rastros de feromona). Esta información sirve para guiar la búsqueda de las otras hormigas de la colonia en el futuro.

2.4.3 Algoritmos Evolutivos

Los algoritmos evolutivos son un esquema de representación que aplica una técnica de búsqueda de soluciones enfocada a problemas de optimización, inspirada en la teoría de la evolución de Charles Darwin. Se basa en el algoritmo de selección propio de la naturaleza, con la esperanza de que así se consigan éxitos similares, en relación a la capacidad de adaptación a un amplio número de ambientes diferentes [2]. En los organismos biológicos, la información hereditaria es pasada a través de los cromosomas que contienen la información de todos esos factores, es decir, los genes, los cuales a su vez están compuestos por un determinado número de valores (alelos). Varios organismos se agrupan formando una población, y aquellos que mejor se adaptan son los que más probabilidad tienen de sobrevivir y reproducirse. Algunos de los sobrevivientes son seleccionados por la naturaleza para ser cruzados y así producir una nueva generación de organismos. Esporádicamente, los genes de un cromosoma pueden sufrir ligeros cambios (las denominadas mutaciones), estos cambios se dan en la naturaleza al azar. En la inteligencia artificial, se simula el comportamiento de los individuos sobre la teoría de la evolución en la metaheurística de Algoritmos Genéticos y Algoritmos Evolutivos. Ambos utilizan la teoría de la evolución con factores tales como población, individuos, operadores genéticos, entre otras.

La diferencia de ambos radica, principalmente, en la representación del individuo en la población. Estos pueden ser representados en forma binaria para el caso de algoritmos genéticos y representación no binaria para algoritmos evolutivos. En ambos algoritmos la forma de utilizar la teoría de la evolución se basa sobre los siguientes conceptos [2]:

- Población: Este es el conjunto de individuos o cromosomas que se utilizan en el problema (los padres potenciales de una nueva población).

- Individuos: Estos son los que conforman la población del problema.

- Fitness: Es el valor que utiliza el algoritmo evolutivo para clasificar la aptitud de los individuos ante el operador de selección.

Operadores Genéticos:

- Recombinación: Este consiste en el intercambio de material genético entre dos cromosomas. Para aplicar la recombinación se escogen aleatoriamente dos miembros de la población.

- Selección: Consiste en elegir la población de la siguiente generación. Lo que se busca es incrementar la probabilidad de reproducir miembros que tengan buenos valores de la función objetivo. Una forma de hacerlos es la denominada Rueda Ruleta (Roulette Wheel).

- Mutación: Este consiste en la modificación del código genético del cromosoma, para aplicar la mutación se establece una frecuencia de mutación.

2.4.4 Algoritmos Voraces (GRASP)

Los métodos GRASP, fueron desarrollados al final de los 80 con el objetivo inicial de resolver problemas de cubrimiento de conjuntos. GRASP, es un procedimiento iterativo en donde cada paso consiste en una fase de construcción y una de mejora. En la fase de construcción se construye una solución tentativa, que luego es mejorada mediante un procedimiento de intercambio hasta que se llega a un óptimo local. En cada iteración del procedimiento constructivo, un elemento es elegido en forma aleatoria de la lista de candidatos para añadirlo a la lista elegida como parte de la solución que se construye. La adición de un elemento a la lista de candidatos se determina mediante una función de tipo devorador2, mientras la selección de un elemento de esa lista de candidatos depende de los que se hayan elegido previamente. Basándose en la construcción eficiente de una solución inicial, puede reducirse el número de pasos necesarios para alcanzar el óptimo local en la fase de mejora. Por lo tanto, el tiempo de cálculo de GRASP depende de diseñar buenos procedimientos de construcción.

2.5 Busqueda Tabú

Los orígenes de la Búsqueda Tabú se ubican a fines de los 60s y principios de los 70s, y se atribuye a Fred Glover, quien desarrolló esta heurística para tratar de resolver problemas de cubierta no lineal (nonlinear covering), aunque varios de sus principios también fueron delineados independientemente por P. Hansen.

La Búsqueda Tabú surgió como un dispositivo que permitiría implementar una estrategia para resolver problemas de programación entera con restricciones sustitutas (surrogate constraints), y aunque su planteamiento original ha sufrido varios cambios a

través de los años, la idea básica propuesta por Glover ha permanecido intacta desde sus orígenes.

La Búsqueda Tabú puede verse como una metaheurística que se superpone a una técnica de búsqueda y que se encarga de evitar que dicha técnica caiga en óptimos locales prohibiendo (o, en un sentido más general, penalizando) ciertos movimientos. El propósito de clasificar ciertos movimientos como prohibidos (Tabú) es para evitar que se caiga en ciclos durante la búsqueda. Debido a esto, Glover sugiere como nombre alternativo para su método el de búsqueda con inhibición débil, ya que los movimientos que se consideran prohibidos constituyen generalmente una pequeña fracción del total de movimientos disponibles, y un movimiento pierde su status de prohibido después de un período de tiempo relativamente corto, volviéndose después nuevamente accesible.

Desde la perspectiva de la Inteligencia Artificial, la Búsqueda Tabú trata de emular (de manera somera) el comportamiento de una persona. Es bien sabido que los humanos poseemos un avanzado mecanismo de intuición que nos permite operar aún con información mínima o nula, pero por lo general solemos introducir un elemento aleatorio (probabilístico) en dichas decisiones, lo cual promueve un cierto nivel de inconsistencia en nuestro comportamiento. La tendencia resultante en estos casos suele desviarnos de una cierta trayectoria preestablecida, lo cual algunas veces puede ser una fuente de errores, pero en otras ocasiones puede llevarnos a una solución mejor.

La Búsqueda Tabú intenta emular este mecanismo fundamental de la ingenuidad humana, pero sin utilizar elementos aleatorios, sino más bien asumiendo que no hay razón para escoger un movimiento que nos lleve a una peor solución, a menos que estemos tratando de evitar recorrer una ruta que ya se examinó previamente. Con esta sola excepción, la técnica buscará, en la mayoría de las veces, el mejor movimiento posible de acuerdo como este definido y utilizado por el problema en cuestión. Esto hace que la técnica se dirija inicialmente de forma directa hacia un óptimo local, pero eso no importa porque la búsqueda no se detendrá ahí, sino que se reinicializará manteniendo la capacidad inicial de identificación del mejor movimiento posible.

Además, se mantiene información referente a los movimientos más recientes en una o más listas tabú, a fin de evitar que una cierta trayectoria previamente recorrida se repita, aunque esta prohibición es generalmente condicional y no absoluta, como veremos más adelante.

La filosofía de la Búsqueda Tabú es derivar y explotar una colección de estrategias inteligentes para la resolución de problemas, basadas en procedimientos de aprendizaje.

2.5.1 Definición de Vecindario Criterio de Intensificación

Los métodos tabú operan bajo el supuesto que se puede construir un entorno para identificar soluciones adyacentes que puedan ser alcanzadas desde una solución actual los intercambios por pares o de un son frecuentemente usados para definir entornos en

problemas de permutaciones. Identificando movimientos que conducen de una solución a la siguiente [6].

Asociado a cada intercambio de posiciones hay un valor del movimiento que se representa en la función objetivo como resultado del movimiento propuesto.

Una representación conveniente de búsqueda en el entorno (vecinos) identifica, para cada solución $x \in X$, un conjunto asociado de vecinos, $N(x) \subset X$, llamado entorno de x. En Búsqueda Tabú, los entornos normalmente se asumen simétricos, es decir, x' es un vecino de x sí y sólo si x es un vecino de x'.

Se puede realizar un método constructivo mediante la estipulación de que X se expande para incluir matrices x cuyas componentes toman valores nulos (no asignados), y mediante la estipulación de que un vecino x' de x puede ser obtenido reemplazando una componente nula de x con una componente no nula.

La decisión de vecino que en particular se define para este problema establece que un vecino de x es cualquier solución que se logre a partir de una solución inicial, intercambiando dos componentes (cursos) de posición que sean viables o intercambiando un curso asignándolo en una posición que sea nula (que no tiene ningún curso asignado).

Vale la pena destacar que intercambiar x por x' es lo mismo que intercambiar x' por x con esto buscamos remarcar que el movimiento es el mismo para efectos de Búsqueda Tabú

$C1$	x	$C5$	.
x	$C4$	$C6$	.
$C2$	x	x	.
$C3$	x	$C7$	Cn

Figura 1 Definición de Vecinos

Para ilustrar lo anterior un vecino de la solución planteada como ejemplo puede ser cualquier intercambio de un curso (C_{ij}) asignado por cualquier otro curso (C_{ij}) distinto o por alguna de las casillas (horarios) que no están asignados (siempre y cuando las restricciones definidas lo permitan). Y cambiar por ejemplo C1 por C4 es lo mismo que cambiar C4 por C1

Criterio de intensificación

A partir de lo anterior se define el criterio de intensificación mediante el cual buscaremos en una lista de posibles movimientos o soluciones vecinas, la que nos producirá la mejora más significativa en la función objetivo y es la que elegiremos como nuestra nueva solución optima local del problema.

En el caso particular del problema que estamos revisando la lista de candidatos posibles será de tamaño siete. De esta lista se escogerá la mejor solución con un $X\%$ de probabilidad / $0 < X < 1$ y cualquiera de la lista con un $1 - X\%$ de probabilidad.

Esto se hace con el fin y bajo el supuesto que en ciertas ocasiones el realizar una desmejora en la solución nos puede llevar en un cierto número de iteraciones a obtener un mejor resultado del que teníamos cuando se realizo la elección del intercambio de posiciones a efectuarse.

2.5.2 Memoria de Corto Plazo

La memoria a corto plazo de la Búsqueda Tabú constituye una forma de exploración agresiva que intenta realizar el mejor movimiento posible sujeto a las restricciones impuestas por el problema [6]. Su funcionamiento básico se ilustra más detalladamente en la Figura.

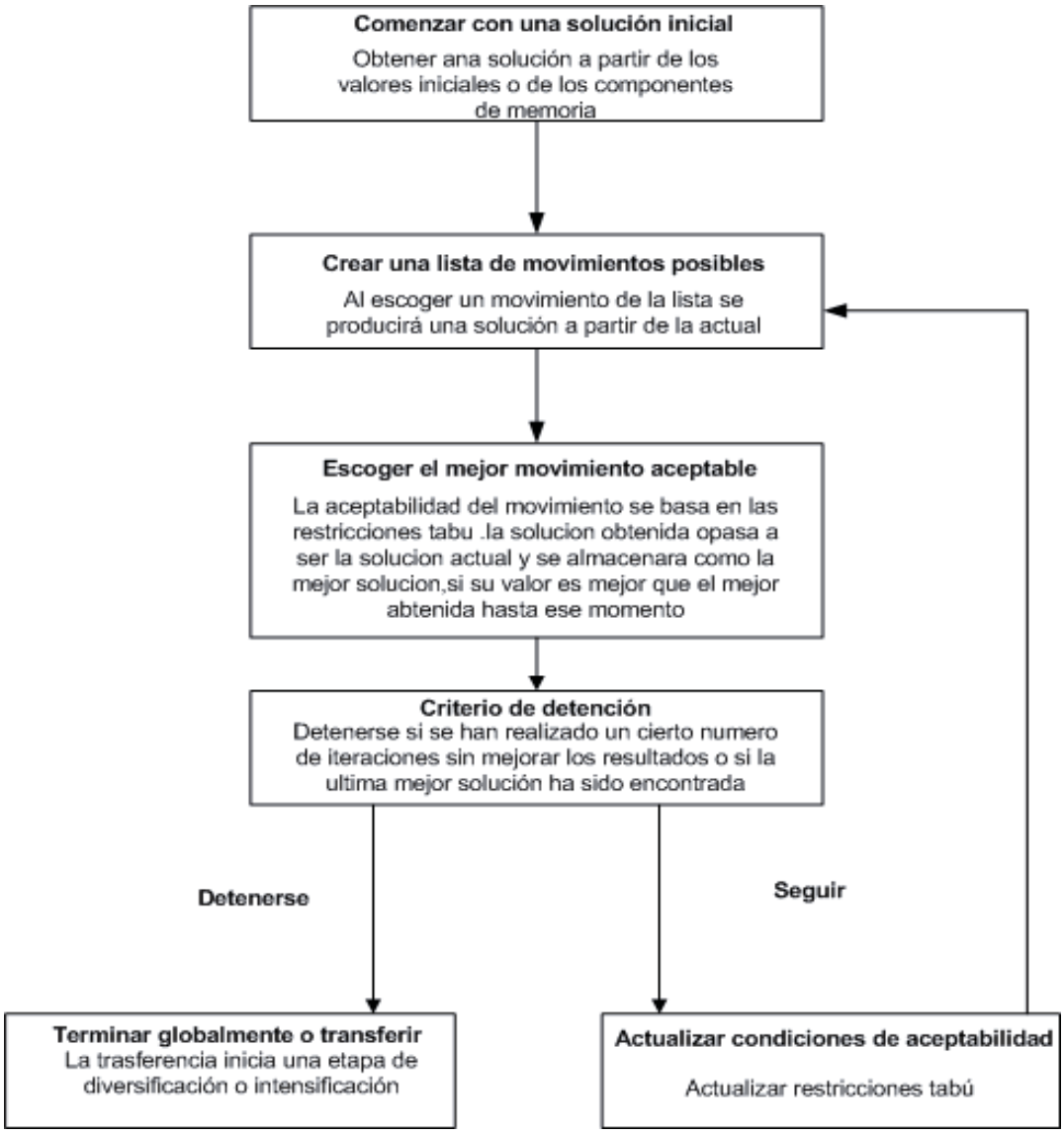


Figura 2 Componente de Memoria a Corto Plazo

2.5.3 Determinación del Mejor Movimiento

Un paso crítico involucrado en la orientación agresiva de la memoria de corto plazo es la selección del mejor movimiento posible (con un $X\%$ de probabilidad / $0 < X < 1$) o cualquier movimiento de la lista (con un $1 - X\%$ de probabilidad) desde un punto cualquiera [6]. La definición del valor de la variable X dependerá del ajuste de parámetros que se realicen al algoritmo. La Figura ilustra esquemáticamente este mecanismo.

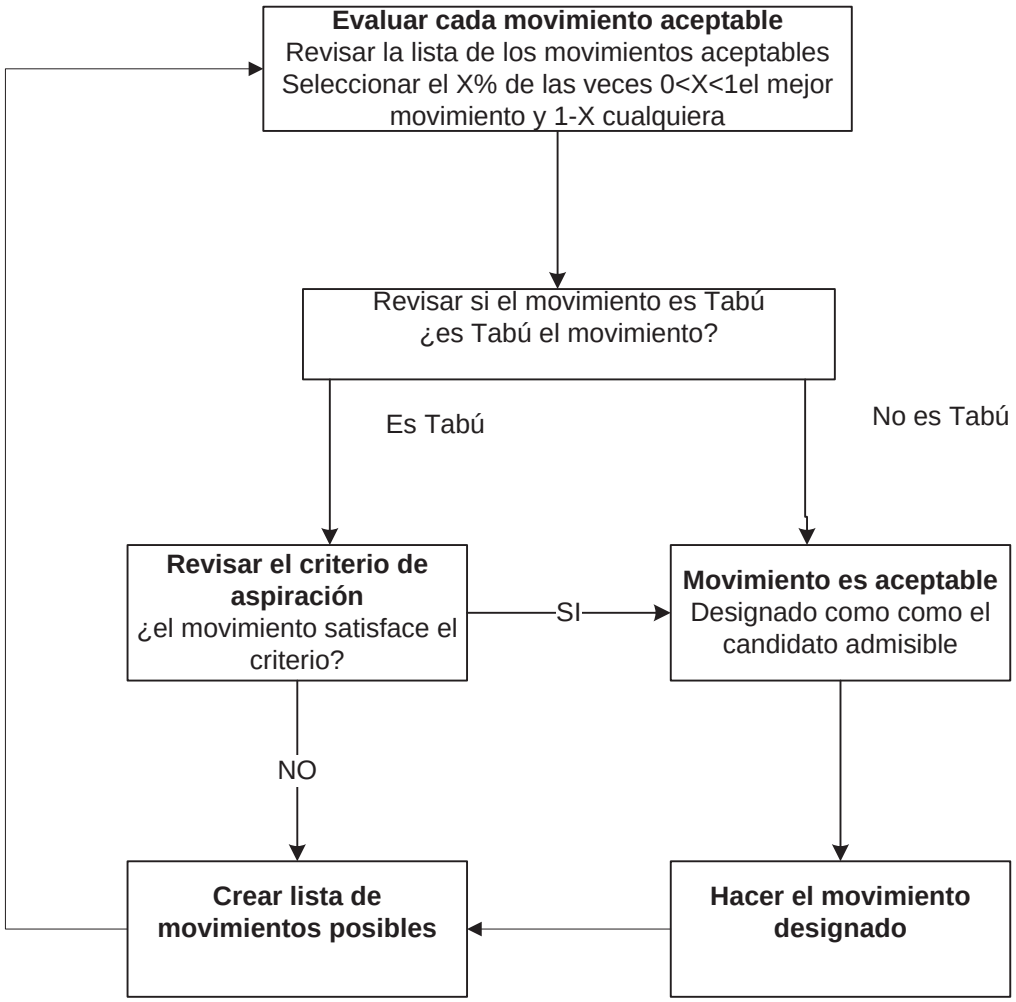


Figura 3 Elección del Movimiento a Realizar

2.5.4 Definición de Movimiento Tabú

El mecanismo principal para explotar la memoria en la Búsqueda Tabú es clasificar un subconjunto de movimientos en un entorno como prohibidos (Tabú). La clasificación depende de la historia de la búsqueda, determinada mediante lo reciente o frecuente que ciertos movimientos o componentes de soluciones, llamados atributos, han participado en la generación de soluciones pasadas. Por ejemplo, un atributo de un movimiento es la identidad del par de elementos que cambian posiciones (en este caso, los dos cursos intercambiados). Como base para evitar la búsqueda desde combinaciones de intercambio repetidas usadas en el pasado reciente, invirtiendo potencialmente los efectos de movimientos anteriores por intercambios que podrían devolver a posiciones previas, clasificaremos como tabú todos los intercambios compuestos por cualquiera de los pares de cursos más recientes; en este caso, para propósitos ilustrativos, los **n** cursos más

recientemente intercambiado. Esto significa que un par de cursos será tabú por una duración (periodo) de **n** iteraciones.

Para este problema se definirán los estados tabú de los movimientos con una duración de **n** (el valor de n está sujeto al ajuste de parámetros) iteraciones decrementándose el valor del tabú en uno por cada iteración que se realice y manteniéndose el movimiento como tabú hasta que su valor llegue a cero nuevamente

Curso	I	J	Iteraciones Tabú
1	-	-	0
2	1	5	2
…	…	…	…
q	4	1	n

Tabla 1 Representación de los Estados Tabú

La tabla muestra los movimientos que se encuentran como tabú que son todos aquellos que su valor es mayor a cero mostrando cual es el numero de iteraciones que les queda como tabú a aquellos movimientos. Las casillas que su valor es cero no son tabú y son candidatos a ser intercambiados.

2.5.5 Criterio de Aspiración

Este criterio se como "si un movimiento produce una solución mejor que cualquier otra obtenida hasta el momento lo efectuamos aunque sea tabú" [7]. Con la definición anterior se pretende precisar que en ocasiones van a existir listas de movimientos posibles en los cuales ninguno de éstos produciría una mejora en la función objetivo.

Los criterios de aspiración se introducen en la Búsqueda Tabú para determinar cuándo pueden ser reemplazadas las restricciones tabú, eliminando así una clasificación tabú aplicada a un movimiento anteriormente. El uso apropiado de este criterio puede ser muy importante para permitir a un método de Búsqueda Tabú alcanzar sus mejores niveles de ejecución. Las primeras aplicaciones empleaban sólo un tipo simple de criterio de aspiración, consistiendo en eliminar una clasificación tabú de un movimiento cuando conduce a una solución mejor que la mejor obtenida hasta ahora.

En se puede realizar un movimiento considerado como Tabú siempre y cuando genere una solución al problema mejor de la que se tiene hasta ahora o en su defecto que el movimiento taba tenga una mejor evaluación con respecto a la funciona objetivo que cualquier movimiento que se encuentra en la lista de posibilidades.

2.5.6 Criterio de Diversificación

Con el criterio de diversificación buscamos hacer búsquedas que estén fuera de los óptimos locales que vamos encontrando a medida que hacemos iteraciones de la Búsqueda Tabú con esto se pretende dirigir la búsqueda a zonas que aun no han sido explotadas.

En el caso particular que se está estudiando, el criterio de diversificación se definirá cada vez que el algoritmo se estanque en el tiempo, que en un cierto número de iteraciones la función objetivo no mejore, se buscará explotar nuevas zonas haciendo que el algoritmo se reinicie a través de una nueva solución inicial obtenida mediante el mismo algoritmo con el cual se dio inicio al proceso.

Para esto se creara una lista de posibles configuraciones iníciales que en este caso serán tres y de estas se elegirá la que presente una evaluación mejor de la función objetivo con un $Y\%$ de probabilidad / $0<Y<1$, y cualquiera de las tres con un $1-Y\%$ de probabilidad de ocurrencia. La elección de los valores que puede tomar la variable Y dependerá del ajuste de parámetros necesarios en el algoritmo.

Lo anterior se explica bajo la premisa que estas soluciones se encuentran en un barrio donde hay buenas soluciones al problema y que en ocasiones se pueden encontrar mejores soluciones si se parte de una que no lo es tanto y para darle un cierto grado de aleatoriedad al proceso.

CAPÍTULO 3:
Análisis de Requerimientos

Los requerimientos que a continuación se especificarán son las funcionalidades que el sistema deberá entregar como mínimo para ser aceptado por el cliente y son estrictamente necesarios para que el software funcione de buena manera y pueda satisfacer las necesidades por las cuales fue necesaria su implementación. Existen dos tipos de requerimientos que se nombraran a continuación; requerimientos funcionales y no funcionales, ambos de gran importancias para que el sistema tenga un óptimo funcionamiento.

3.1 Requerimientos Funcionales

- El sistema debe entregar un horario clases semanal válido para la Escuela de Ingeniería Informática de la PUCV.
- El sistema debe entregar el horario semanal de cada profesor de la Escuela de Ingeniería Informática de la PUCV.
- El sistema debe entregar un listado de las salas y las horas que se encuentran ocupadas o disponibles en las distintas claves horarias.
- El sistema debe registrar los datos necesarios de los profesores, salas disponibles y cursos que dicta la escuela para que el sistema pueda entrar en funcionamiento.
- Es sistema debe aceptar cambios en los datos que se han ingresado con anterioridad.
- El sistema debe permitir la navegación por los distintos módulos que lo integran.
- El sistema debe permitir modificar la solución entregada.

3.2 Requerimientos No Funcionales

- El sistema debe aceptar sólo datos válidos.
- El sistema debe entregar la solución de y los datos de salida de una manera comprensible.
- El sistema debe presentar mensajes de error en caso que algo irregular ocurra.
- El sistema debe garantizar que no se modifiquen los datos mientras se está calculando una solución

CAPÍTULO 4:
Metodología de Trabajo

Con el objetivo de elegir una metodología para la realización de nuestro proyecto, realizaremos a continuación una revisión y comparación de algunos de los métodos existentes. Consideraremos metodologías tanto tradicionales como ágiles para luego seleccionar la que, a nuestro juicio, se ajusta de mejor manera a las características del proyecto de asignación de cursos.
Se compararán las siguientes metodologías:

Tradicionales:

- Cascada
- Construcción de Prototipos
- UP (Unified Process)

Ágiles:

- Extreme Programming (XP)
- AUP (UP Ágil)

4.1 Metodologias Tradicionales

Se hará una revisión de las metodologías de desarrollo más utilizadas en la actualidad al momento de desarrollar un software.

4.1.1 Construcción de Prototipos

La construcción de prototipos consiste en desarrollar una versión inicial del producto sin implementar completamente la funcionalidad del sistema, dentro de la construcción de prototipos se puede destacar los desechables y los evolutivos los primeros entregaran una visión general inicial del producto y solo servirá para captar requerimientos para luego, como su nombre lo dice, desecharlos. El segundo tipo de prototipos los evolutivos además de usarse para captar requerimientos no se desechara sino que se utilizará como una versión entregable del producto y que en cada iteración se entregará una versión con más funcionalidades (incrementos sucesivos) hasta llegar a una versión final la cual sería el producto terminado [8].

Los prototipos son una buena alternativa en el desarrollo de proyectos en el cual se tiene completo conocimiento y dominio del problema, de las técnicas que existen para dar solución a este y en equipos de desarrollo en los cuales se tenga considerada una gran

cantidad de tiempo destinado al desarrollo. Debido a lo anterior ya que en particular por el tiempo disponible de los desarrolladores y que la técnica mediante la cual se pretende resolver el problema necesita ser dominada de antemano, sumado a que la primera fase del proyecto (construcción del algoritmo para asignación de cursos) no dispone de una interfaz para la cual crear prototipos. Este método se haría inviable para el desarrollo del software.

4.1.2 Cascada

El modelo de cascada define el desarrollo del proyecto en forma lineal de pasos donde cada una de sus etapas está bien definida con hitos claramente establecidos y donde cada paso se realiza de forma estricta después del que lo precede. Esto hace que el método no sea muy flexible, por lo que si se omite algún requerimiento o detalle en alguna etapa inicial, esto repercutiría en un gran costo para el proyecto ya que se haría necesario rehacer todo el trabajo desde el punto en que se omitió aquel requerimiento [9].

Debido a aquella inflexibilidad del método de desarrollo y a que al inicio del proyecto pueden no estar bien establecidos los límites de este y que se pueden omitir de parte nuestra algún requerimiento o detalle de la técnica de resolución a utilizar y que lo anterior puede pasar en más de una ocasión, no pensamos que este método se adapte de buena forma al desarrollo de este proyecto en particular.

4.1.3 Proceso Unificado (UP)

El proceso unificado es una forma disciplinada de asignar tareas y responsabilidades a un equipo de desarrollo y asegurar la producción de un software de calidad dentro de los plazos y presupuestos establecidos. UP también nos provee de una guía para utilizar UML que es un lenguaje completo para el modelamiento del software[10].

También recoge características de desarrollo de software lo sé que define como “mejores prácticas” dentro de estas características se destacan un desarrollo iterativo el cual nos permite una comprensión creciente de los requerimientos a medida que se va desarrollando el software, abordando de manera temprana los aspectos y tareas más riesgosas, la administración de los requerimientos que nos promociona una guía para obtener los requerimientos, organizarlos, documentarlos y definir sus restricciones, rastrear y documentar decisiones., entre otras características que UP nos ofrece. Además este modelo se enfoca a la calidad lo que proporciona un marco de trabajo en el cual no solo se estará elaborando la solución del sistema sino que también se está asegurando que el producto cumpla con lo requerido. Este método en particular permite una constante mejora del producto al admitir el desarrollo de prototipos, el análisis constante requerimientos, proporciona un desarrollo iterativo donde cada una de las iteraciones se ve como una pequeña cascada con todas sus etapas.

4.2 Metodologias Ágiles

Las siguientes metodologías son relativamente nuevas para el desarrollo de software y se alejan de las tradicionales poniendo más énfasis en el desarrollo mismo del software que en la producción de documentación.

4.2.1 Extreme Programming

Extreme Programming o XP como también se le conoce, es una de las metodologías ágiles más populares, que se usa en proyectos con alto nivel de riesgos y requerimientos dinámicos enfatiza la cercanía con el cliente y promueve el trabajo en equipo, consta de reglas y practicas simples que se apoyan unas a otras [11].

Los equipos de trabajo se mantienen a un mínimo y en proyectos grandes el personal se reparte en varios equipos que trabajaran cada una en un componente. Enfatiza las reglas de programación y los estándares de codificación en cuanto a la legibilidad del código.

4.2.2 AUP

AUP es una metodología ágil basada en UP. Describe un acercamiento para el desarrollo de aplicaciones de negocios usando técnicas ágiles y que pretende eliminar algunas de las limitaciones de éste haciendo más flexible el proceso e introduciendo un enfoque basado no tanto en la teoría sino que más en la práctica y aún así manteniéndose fiel a la naturaleza del proceso unificado UP [12].

Método Elegido

Luego de conocer las ventajas y desventajas de estas metodologías hemos optado por elegir el proceso unificado ágil AUP debido a que, según nuestro punto de vista, nos entrega lo mejor de los recursos de UP sumado a la flexibilidad y el pragmatismo en la visión del proceso que se logra a través de las metodologías ágiles.

4.3 Plan de Trabajo

Para realizar el siguiente proyecto se ha decidido seguir la metodología AUP, debido a que nos permite llevar a cabo una planificación del proyecto mucho más detallada y específica, permitiéndonos controlar los procesos de nuestro proyecto además de proveernos herramientas con las cuales podremos ser más ordenados respecto a las tareas que se deben realizar en el proyecto.

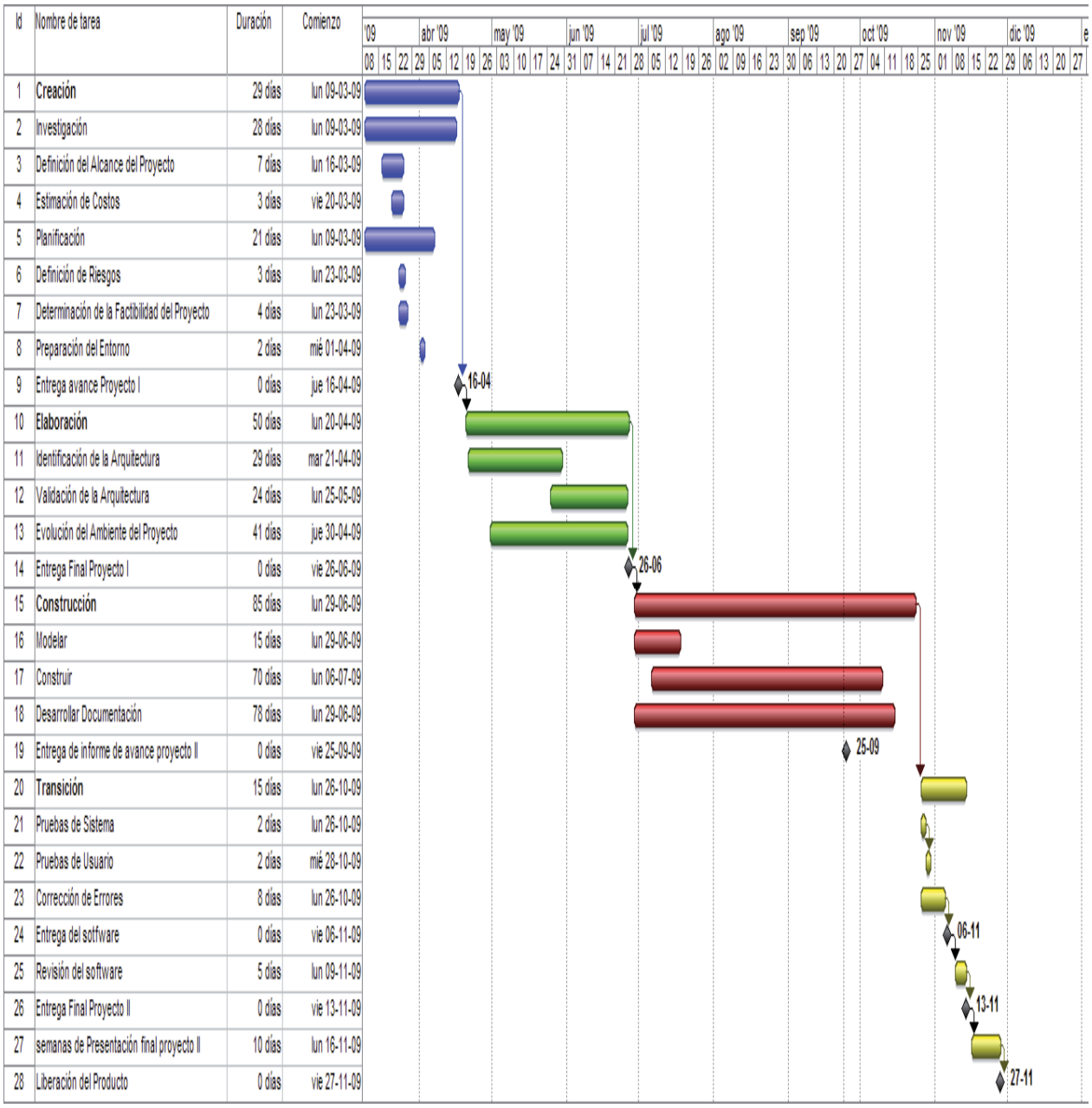


Figura 4 Planificación del Proyecto: Carta Gantt

CAPÍTULO 5:
Viabilidad y Modelado del Proyecto

En el capítulo siguiente se hará una revisión desde los distintos puntos que pueden afectar el proyecto en su desarrollo se revisaran los estudios de factibilidad (técnica, legal, operativa, etc.) se igual forma se revisara y se hará un análisis de los posibles riesgos que puedan afectar al proyecto y por último se efectuará el modelado matemático del problema.

5.1 Estudio de Factivilidad

En este proyecto e software como en cualquier otro proyecto debemos realizar un análisis previo considerando varios aspectos que harán posible o no la realización del mismo entre estos aspectos revisaremos si es posible su realización desde el punto de vista económico, legal, técnico y operativo, estos estudios se realizan debido a que los recursos disponibles para realizar el proyecto son limitados o están sujetos a una serie de restricciones que lo pueden hacer inviable el mismo.
En la sección siguiente daremos una revisión a cada uno de estos puntos en forma detallado.

5.1.1 Factibilidad Técnica

Se debe poner gran énfasis en si se encuentran disponibles los recursos técnicos necesarios para la realización del proyecto. Entre estos recursos se encuentran la disponibilidad de software, hardware, tecnología y recursos humanos necesarios.

Durante la primera etapa del proyecto en la cual se realizara la investigación necesaria de las técnicas existentes para resolver el problema y seleccionar la que se va a utilizar la Escuela de Ingeniería Informática pone a disposición nuestra los laboratorios (de lunes a viernes en jornada completa y el sábado medio día) en los cuales podemos encontrar computadores conectados a Internet lo cual facilita nuestra investigación ya que nos permiten acceder a los papers necesarios para la investigación. Y en la segunda etapa donde se desarrollara el software también podemos hacer uso de las computadoras necesarias.

En cuanto al desarrollo del software se utilizarán tecnologías libres como son JAVA y C, el Modelamiento del software se realizará con las herramientas disponibles por la escuela por lo que no se observa ningún impedimento en el uso de estas tecnologías para la realización y desarrollo del proyecto.
Por lo tanto se considera que el proyecto es viable desde el punto de vista técnico.

5.1.2 Factibilidad Económica

Debido al carácter del proyecto enfocado a la investigación, y no al uso empresarial sino que al uso de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso, la tasa de retorno del capital no se considerará como un punto para tomar una decisión. Sin embargo existen otros puntos medibles, como son el costo del hardware y el software necesario para desarrollar e implementar el sistema.

Herramienta	Costo
Visio	$251000
Office hogar y estudiantes	$98000
Project	$595000
Total	$944000

Tabla 2 Costo de las Herramientas a Utilizar

Si consideramos que para la ejecución de un prototipo y posterior desarrollo final del software es necesario por lo menos el uso de un computador

Número de equipos	Costo
1	$300.000
Total	$300.000

Tabla 3 Costo de los Equipos a Utilizar

Si consideramos los costos de las herramientas a utilizar y los costos de los equipos la suma haciende a $1244000(un millón doscientos cuarenta y cuatro mil pesos) lo cual no es una suma despreciable pero gracias a que la Escuela de Ingeniería Informática nos proveerá de los equipos necesarios para la realización del proyecto los gastos obligatorios por este concepto desaparecerán y lo gastos que provienen de las herramientas necesarias la universidad también posee las licencias, el costo final del proyecto tiene un valor de $0 (cero pesos).
Por lo tanto se considera que el proyecto es viable desde el punto de vista económico.

5.1.3 Factibilidad Legal

Si tenemos en cuenta que las herramientas a utilizar para el desarrollo del proyecto son de uso libre (JAVA y C) por lo que no hay que pagar ningún tipo de licencia y si agregamos a esto que el desarrollo de este software no tiene por propósito ninguna actividad comercial de parte nuestra (obtener retribuciones económicas con su desarrollo) no se observa ningún impedimento de carácter legal que nos impida seguir adelante con el proyecto. Con respecto a la plataforma donde se utilizara el software se hará uso de las herramientas que la Escuela de Ingeniería Informática nos provee y que se encuentran con sus licencias al día para el uso de los estudiantes.

Por lo tanto se considera que el proyecto es viable desde el punto de vista legal.

5.1.4 Factibilidad Operativa

Debido a que los integrantes del equipo de desarrollo tienen conocimiento y dominio sobre la mayoría de las herramientas o tecnologías con las que se pretende desarrollar el producto (JAVA y C) y que la técnica con la que se planea resolver el problema está siendo estudiada y asimilada de buena forma no vemos impedimentos de tipo operativo que nos limiten para poder seguir adelante con el proyecto y lograr obtener buenos resultados en su desarrollo.

Por lo tanto se considera que el proyecto es viable desde el punto de vista operativo.

5.2 Análisis de Riesgos

Mantener los riesgos identificados permitirá darse cuenta cuando se esté frente a alguno de ellos y decidir cuál es mejor camino o acción a seguir con el fin de minimizar o eliminar las condiciones adversas a las que pueden llevar esos riesgos a través de los planes de contingencia que se pueden desarrollar para cada uno de los riesgos identificables para poder hacer un mejor uso de los recursos que se encuentran disponibles y que se pueden ver afectados en caso de que ocurra alguno de los riegos y a través de esto afectar el desarrollo del proyecto.

De lo anterior se puede desprender que un riesgo es cualquier situación adversa para el desarrollo del producto. A cada riesgo se le asignará una probabilidad de ocurrencia (alta, moderada, baja) y el efecto (catastrófico, serio, tolerable, insignificante) que puede tener sobre al proyecto. Para cada uno de los riesgos identificados se tendrá un plan de mitigación y para los riesgos que tengan una alta probabilidad de ocurrencia y afecten al proyecto de forma seria o catastrófica se tendrá un plan de contingencia asociado.

5.2.1 Clasificación de los Riesgos

- **Riesgos del proyecto**, amenazan la planificación del proyecto, afectan la calendarización o los recursos, por ello conllevan un retraso en los tiempos y un aumento en los costos.

- **Riesgos de producto**, afectan la calidad o desempeño del software que se está desarrollando, si los riesgos de este tipo se llegan a hacer realidad, la implementación puede volverse difícil o imposible.

- **Riesgos del negocio**, afectan a la organización que desarrolla o procura el software.

Para administrar los riesgos asociados al proyecto, se efectuaran las siguientes actividades:

Identificación de los riesgos, identificar los riesgos de proyecto, negocio y producto.

- **Análisis de riesgos**, evaluación de la probabilidad y consecuencia de los riesgos.

- **Planificación de riesgos**, elaboración de planes para minimizar o evitar los efectos del riesgo.

- **Monitoreo de riesgos**, monitorear los riesgos durante todo el proyecto.

5.2.2 Identificación de los Riesgos

En la siguiente sección se hará una identificación de los riesgos clasificándolos según su tipo o parte del proyecto que afecta (producto, proyecto, negocio), la probabilidad de ocurrencia y los efectos que puede tener sobre el mismo según el modelo propuesto por Pressman [13].

Descripción del riego	Tipo de riesgo	Probabilidad	Efectos	Identificador
Incumplimiento de los plazos	Producto Proyecto	Alta	Serio	1
Algún integrante del equipo decide retirarse	Negocio	Baja	Catastróficos	2
Los integrantes del grupo de trabajo no dominan o no conocen las tecnologías a utilizar para el diseño y	Proyecto Negocio	Alta	Catastrófico	3

realización del proyecto				
La dimensión del proyecto no se mantiene dentro de los márgenes establecidos	Proyecto	Moderada	Serio	4
No disponibilidad de software o hardware necesario para el producto	Proyecto	Baja	Serio	5
Los hitos de revisión de avance no son cumplidos	Proyecto Producto	Alta	Serio	6
Algún miembro del proyecto pasa a llevar la autoridad del otro e intenta involucrarse en el área del otro en forma poco sana	Negocio	Baja	Insignificante	7
Se realizan movilizaciones estudiantiles durante la realización del proyecto	Proyecto	Alta	Insignificante	8
El tiempo dedicado al proyecto se ve afectado por la realización de otras actividades	Proyecto	Moderada	Serio	9

Tabla 4 Identificación y Análisis de Riesgos

5.2.3 Planes de Mitigación de los Riesgos

Los planes de mitigación están asociados a todos los riesgos, que han sido identificados en el proyecto. De esta forma cada riesgo observado cuenta con un plan para minimizarlo aunque la ocurrencia de dicho riesgo sea baja. Sin embargo, no todos los riesgos cuentan con planes de contingencia, esto debido a que los planes de contingencia tienen un costo asociado, el cual incide en el proyecto, es por ello que estos planes se generan solamente para aquellos casos en los cuales la ocurrencia del riesgo está entre el rango media y alta y además tenga una incidencia critica o catastrófica en el proyecto [13].

Identificador del riesgo	Medidas para minimizar mitigar el riesgo	Plan de contingencia asociado

1	Evaluación y seguimiento de las actividades a desarrollar durante la duración del proyecto	1
2	Reuniones periódicas con el fin de motivar a los integrantes del equipo	-
3	Evaluar periódicamente el avance de los integrantes y cuánto han aprendido	2
4	Establecer los límites del sistema de forma clara	3
5	Solicitud en los plazos establecidos de los equipos necesarios, y obtener las herramientas necesarias con tiempo de holgura.	-
6	Revisiones semanales de los grados de avance en las actividades a cumplir.	4
7	Establecer los cargos y responsabilidades en forma clara y temprana, estableciendo los límites para los miembros del equipo.	-
8	Continuar con la planificación del proyecto, y actualizar los hitos entregables a la Universidad.	-
9	Planificar y dividir equitativamente los tiempos para la responder en cada una de las actividades.	5

Tabla 5 Planes de Mitigación del Proyecto

5.2.4 Planes de Contingencia

En caso que un riesgo con una alta probabilidad de ocurrencia y que afecte al proyecto de forma catastrófica o seria ocurra es necesario tener un plan de contingencia que permita minimizar los efectos que esto puede tener sobre el desarrollo de un proyecto.

Plan de contingencia	Descripción del plan de contingencia
1	Realizar un reajuste de la planificación y de los pasos con el fin de con la fechas fijadas desde un principio

2	Reunirse con personas que dominen las tecnologías a utilizar para que realicen charlas de manera que los integrantes del equipo puedan dominar dichas tecnologías
3	Volver a fijar los límites del sistema
4	Fijar horarios de trabajo fijos, y penalizar el incumplimiento de estos horarios.
5	Establecer horas fijas a cada actividad y velar por el cumplimiento de ellas

Tabla 6 Plan de Contingencia en Caso de Ocurrir un Riesgo

5.3 Modelado del Problema

En esta sección se revisará la manera con la cual se pretende resolver el problema que se ha planteado, para esto se planteará un modelo matemático para la solución las estructuras que se utilizaran y como se representaran estas para llegar a una solución.

5.3.1 Modelado Matemático del Problema

A continuación se presenta el modelo matemático mediante el cual se pretende dar solución al timetabling se especificaran las restricciones tanto obligatorias como opcionales a las que está sujeto el modelo además de la función objetivo con que se evaluarán las posibles soluciones.

C_{ijk} : Costo curso i, sala j

X_{ijk} : Variables binarias de asignación {1,0}

1 si e curso i está asignado a la sala j , 0 en otro caso

i : curso / 1 <= i <= n

j : sala / 1 <= j <= m

k : clave / 1 <= k <= p

f_i : Capacidad de la sala i / S : cantidad de grupos

e_i : Número de estudiantes del curso i

I_r : Cursos dictados por el profesor r

I_s : Cursos a los que asiste un grupo s de estudiantes

T_r : Clases no disponibles para el profesor r

q : Número de profesores

S a:

No debe quedar ningún curso sin asignar

$$\sum_{i=1}^{n} X_{ijk} \leq 1 \qquad 1 \leq i \leq n \;\; , \;\; 1 \leq k \leq p \tag{5.3.1.1}$$

El cupo esperado del cuso no debe ser mayor que la capacidad de la sala

$$e_{ij}\, X_{ijk} \leq f_j \qquad 1 \leq i \leq n \;\; , \;\; 1 \leq j \leq m \;\; , \;\; 1 \leq k \leq p \tag{5.3.1.2}$$

Cada profesor puede dictar hasta un curso por período

$$\sum_{i \in I_r} \sum_{j=1}^{m} X_{ijk} \leq 1 \qquad 1 \leq k \leq p \;\; , \;\; 1 \leq n \leq q \tag{5.3.1.3}$$

Cada profesor puede dictar cursos solo en sus horas disponibles

$$\sum_{i \in I_r} \sum_{j=1}^{m} \sum_{k \in T_r} X_{ijk} = 0 \qquad 1 \leq r \leq q \tag{5.3.1.4}$$

Las claves pertenecientes al mismo grupo de estudiantes no se pueden dictar a la misma hora

$$\sum_{i \in I_s} \sum_{j=1}^{m} X_{ijk} \leq 1 \qquad 1 \leq k \leq p \;\; , \;\; 1 \leq s \leq S \tag{5.3.1.5}$$

No deben quedar clases sin asignar para cada ramo

$$\sum_{j=1}^{m}\sum_{k=1}^{p} X_{ijk} \le R_i \qquad 1 \le i \le n \tag{5.3.1.6}$$

Restricciones Suaves

- Penalizar salas sobredimensionadas para el curso

$f_i - e_i$ = aumentar en 1 el costo de cada asiento extra en la sala

- Penalizar claves horarias (horarias) dentro del mismo día para profesores

D(r , k) = distancia para el profesor r a partir de la clave k

- Considerar ramos no conflictivos para calendarizarlas en horarios distintos

NC (k) = cantidad de horarios no conflictivos en el horario k

Función de costos

- Preferencia de horario para los profesores

C_{ijk} = define costo de asignar el curso k en la clave i en la sala j (prioridad de los profesores)

F.O:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p}\sum_{r=1}^{q}[C_{ijk}\, X_{ijk} + (f_i - e_i) + NC(k))] + \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{r=1}^{q}\sum_{\substack{k=1,8,16\\24,32,40}}^{p=k} D(r,k) \tag{5.3.1.7}$$

CAPÍTULO 6:
Desarrollo del Sistema

En el siguiente capítulo se muestra toda la ingeniería realizada para desarrollar el sistema de asignación de cursos para la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso con el fin de entender de mejor manera como se desarrolla el software.

Para lo anterior se utilizarán los diagramas que pone a disposición UML para reflejar la funcionalidad del sistema en cuanto a su interfaz gráfica y como se presentarán los datos obtenidos. Para representar el módulo que se encarga de realizar el cálculo de solución al problema se utilizarán diagramas de flujo ya que es la forma más sencilla para que se entienda como realiza los cálculos este módulo.

6.1 Diseño del Algoritmo

Los diagramas que a continuación se presentan ilustran la forma en que el sistema calculará la solución en cada uno de los módulos necesarios descritos en capítulos anteriores respecto a la metaheurística (Búsqueda Tabú) mediante la cual se obtienen horarios válidos.

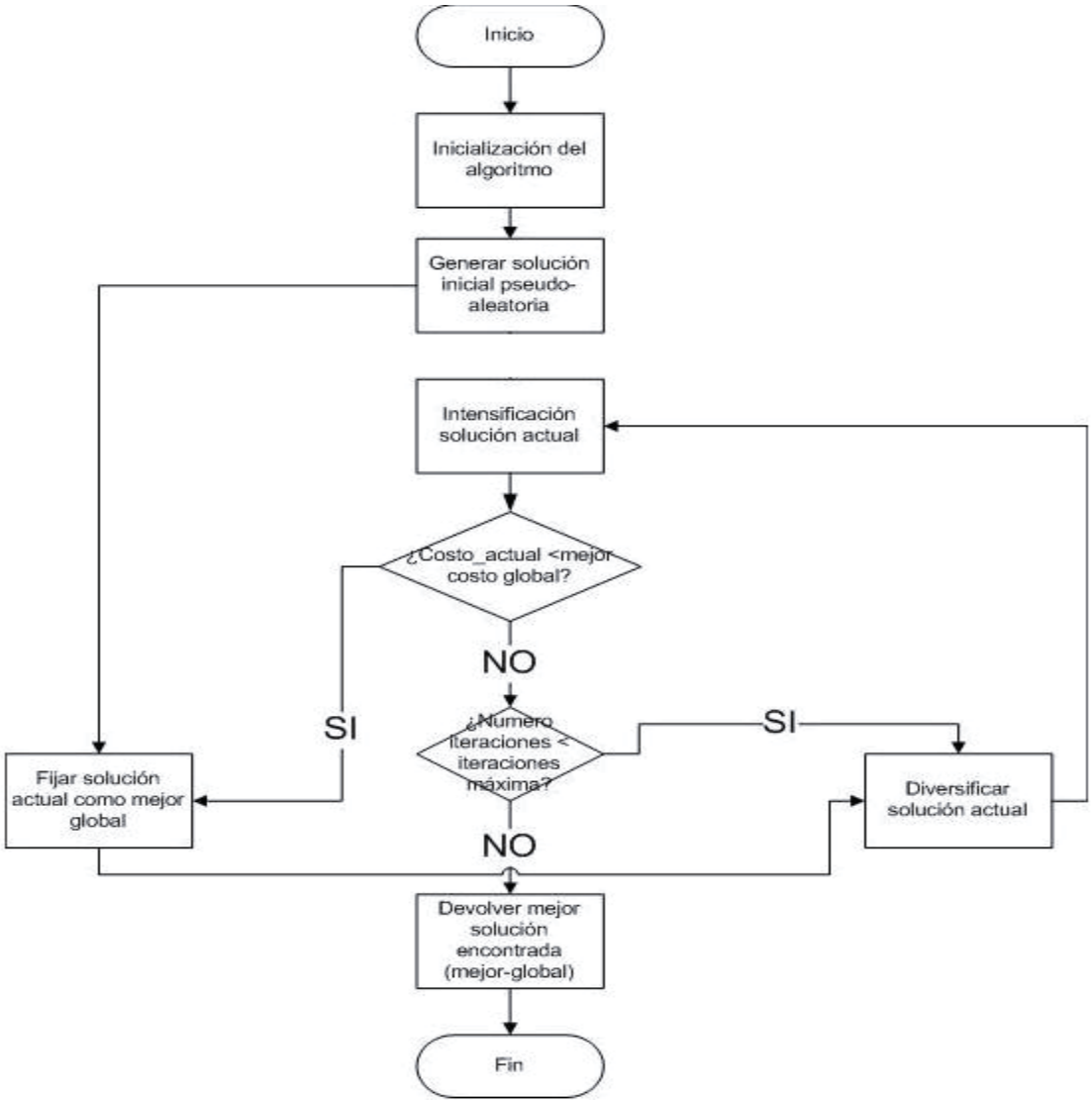


Figura 5 Algoritmo de Asignación Alto Nivel

En el diagrama anterior se muestra la forma general del algoritmo de asignación en donde se pueden apreciar sus componentes más importantes.
Cada vez que el algoritmo se ejecuta, primero debe pasar por una fase de inicialización, en donde las estructuras internas son llenadas con los datos provenientes del usuario (por medio de la interfaz gráfica).

Luego procede a ejecutar una serie de iteraciones principales, que consisten en una diversificación de la solución actual (la solución que sigue la línea de modificaciones exitosas) seguida por una intensificación que se ocupa de extraer el óptimo local de ese conjunto de soluciones posibles.

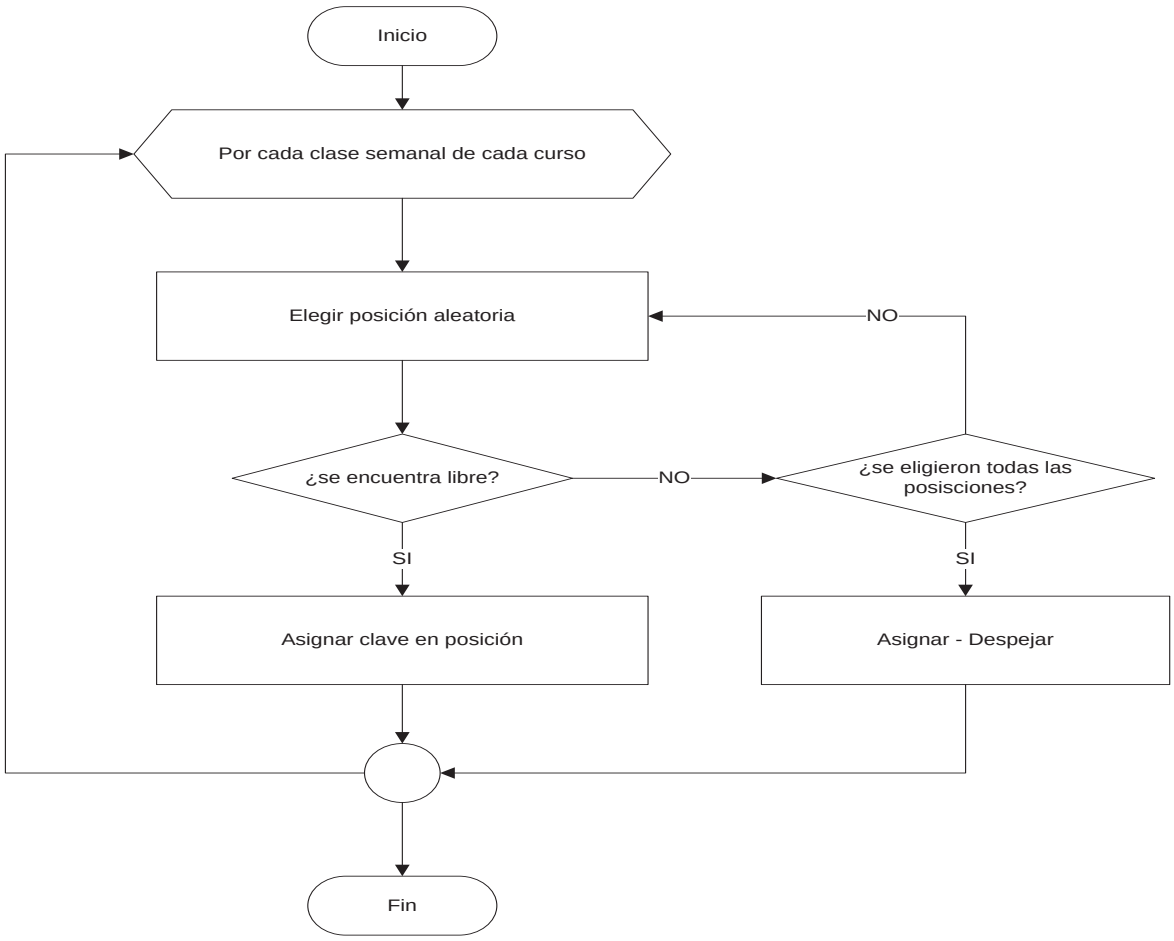


Figura 6 Diagrama de Flujo Solución Inicial

El diagrama solución inicial muestra los pasos necesarios para lograr una configuración horaria válida con la finalidad de ser el punto de inicio para comenzar iterar desde esa configuración hasta obtener una solución final de alta calidad.

La solución inicial consta de repetidas asignaciones aleatorias, donde cada clase semanal de cada curso es acomodada dentro de la matriz de asignaciones. En caso de que todas las opciones de asignación para una clase estén ocupadas se procede a hacer cambios dirigidos (*asignar_despejar*) en la solución para poder acomodar factiblemente todas las clases.

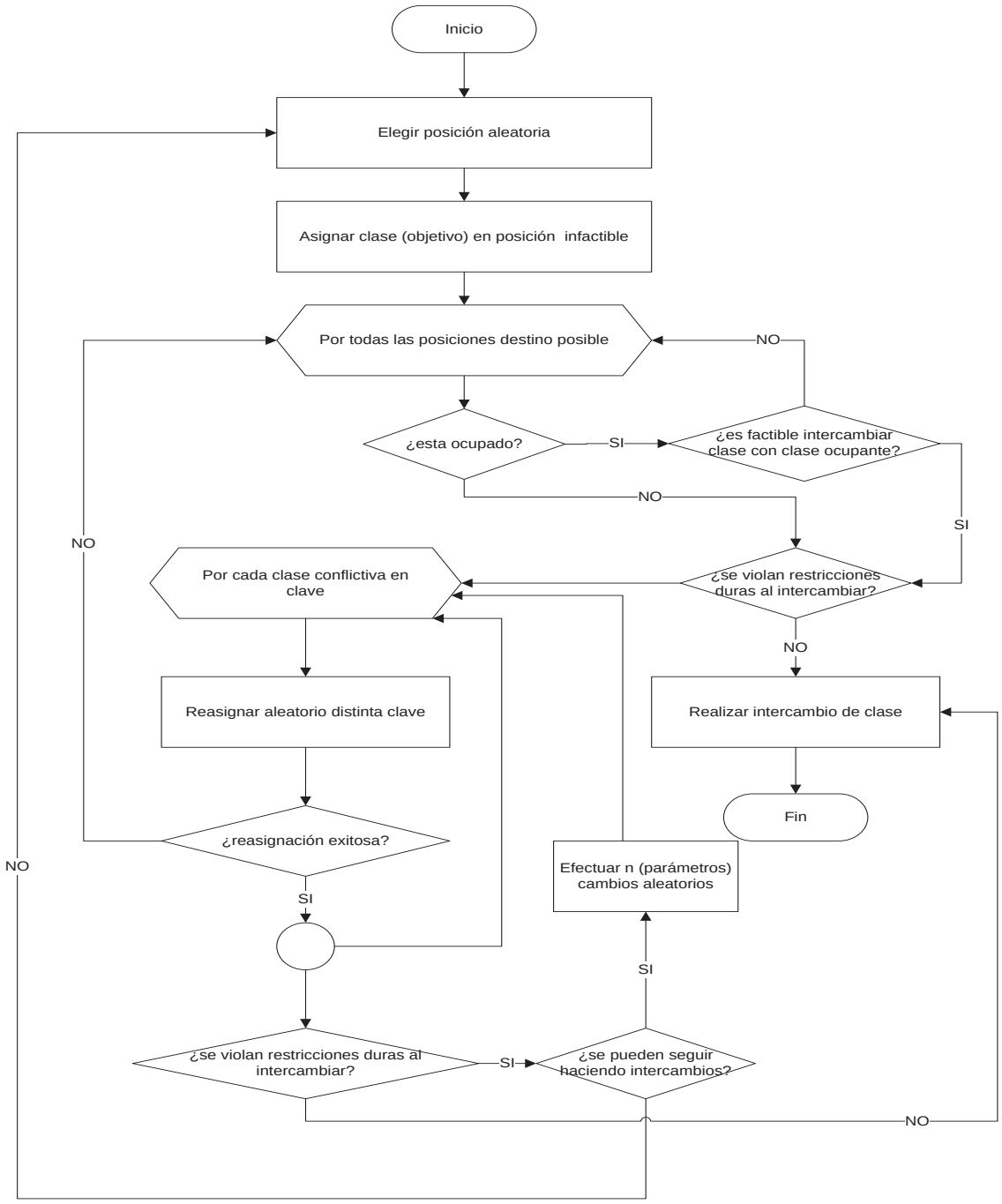


Figura 7 Asignar Despejar

Asignar Despejar se utiliza como parte de la función que genera la solución inicial. Cuando una clase no puede ser asignada directamente en ninguna posición factible, por violar restricciones duras, se invoca a esta función, la que se encarga de hacer intercambios en las claves conflictivas, hasta se torna factible ubicar la clase objetivo. Esto se logra asignando primero la clase objetivo en una posición infactible e intentar una serie

intercambios de asignaciones (flips) factibles, hasta que todas las clases quedan en una posición que no viole restricciones duras.

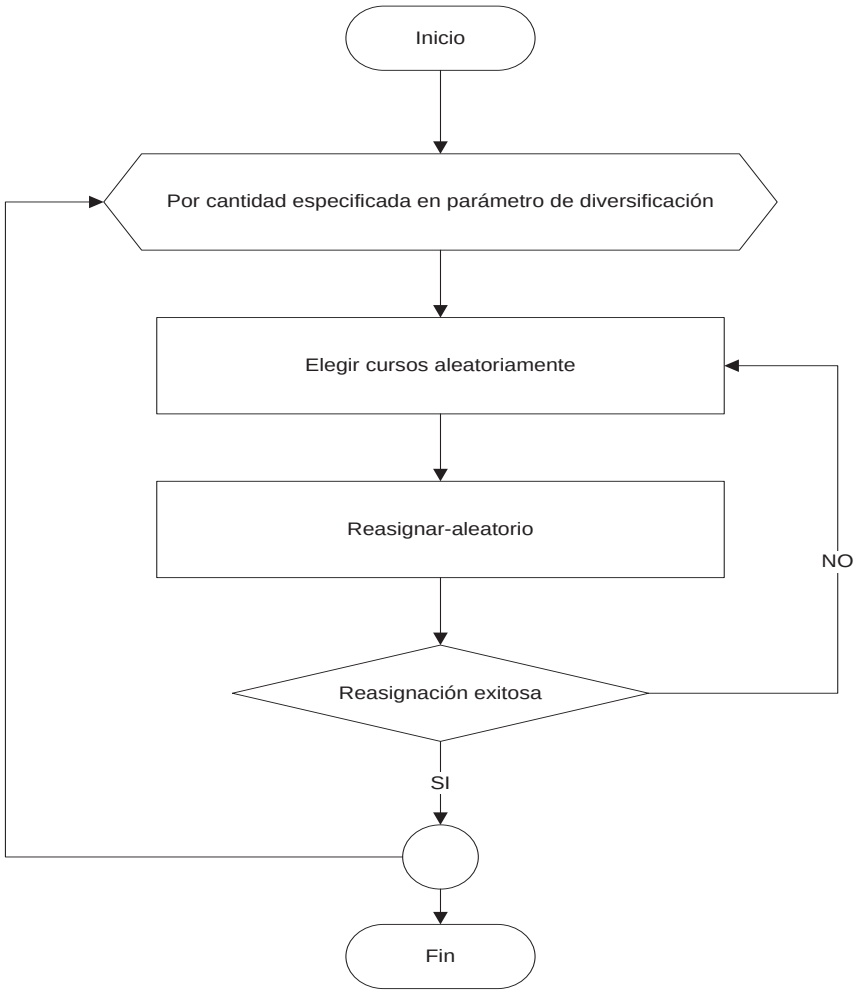


Figura 8 Diagrama de Flujo Diversificar

El algoritmo de diversificación reasignar aleatoriamente una cantidad de clases especificadas por su parámetro correspondiente que a su vez son elegidas al azar, llama a la función '**reasignar_aleatorio**' que se encarga de acomodar un curso (en una iteración del algoritmo diversificar) en una posición aleatoria distinta de donde se encuentra asignado actualmente de tal manera que la factibilidad se mantenga y ningún curso quede sin asignar. Es importante notar también que el algoritmo '**reasignar_aleatorio**' puede fallar su reasignación en determinados casos, al ocurrir esto la función que lo llama, en este caso el algoritmo de diversificación, simplemente elige otra clase para reasignar.

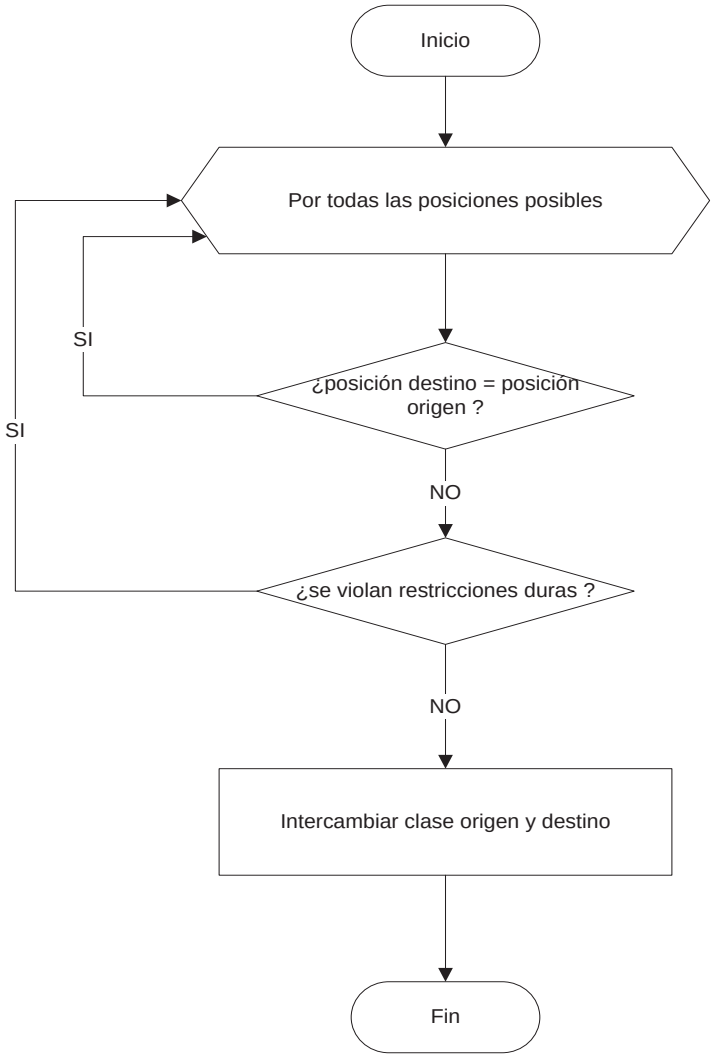


Figura 9 Reasignación Aleatoria

El algoritmo de reasignación aleatoria, llamado ‘reasignar_aleatorio’ es uno de los dos procesos principales dentro del solver tabú, éste se encarga de mover una clase a una posición aleatoria pero factible.

Para realizar esto, es necesario tener varias cosas en cuenta. Primero antes de elegir una clave de asignación se debe primero verificar que la clave que se va a asignar ahí no sea incompatible con las que ya están asignadas, vale decir, otras clases del mismo profesor, si éste es el caso se trata de buscar nuevas posiciones de asignación para todas las clases que estaban haciendo conflicto en esa clase de una manera recursiva ejecutando otra instancia del mismo algoritmo. Nuevamente, para realizar esto es necesario llevar un registro de las clases que están esperando una respuesta (de las iteraciones invocadas para despejar esa clave de conflictos), para que las iteraciones invocadas no vuelvan a asignar clases conflictivas en esos espacios. Lo mismo se aplica en el otro caso que corresponde a cuando no existen (o ya se hizo el despeje) clases conflictivas en la clave pero la clave de destino ya está ocupada.

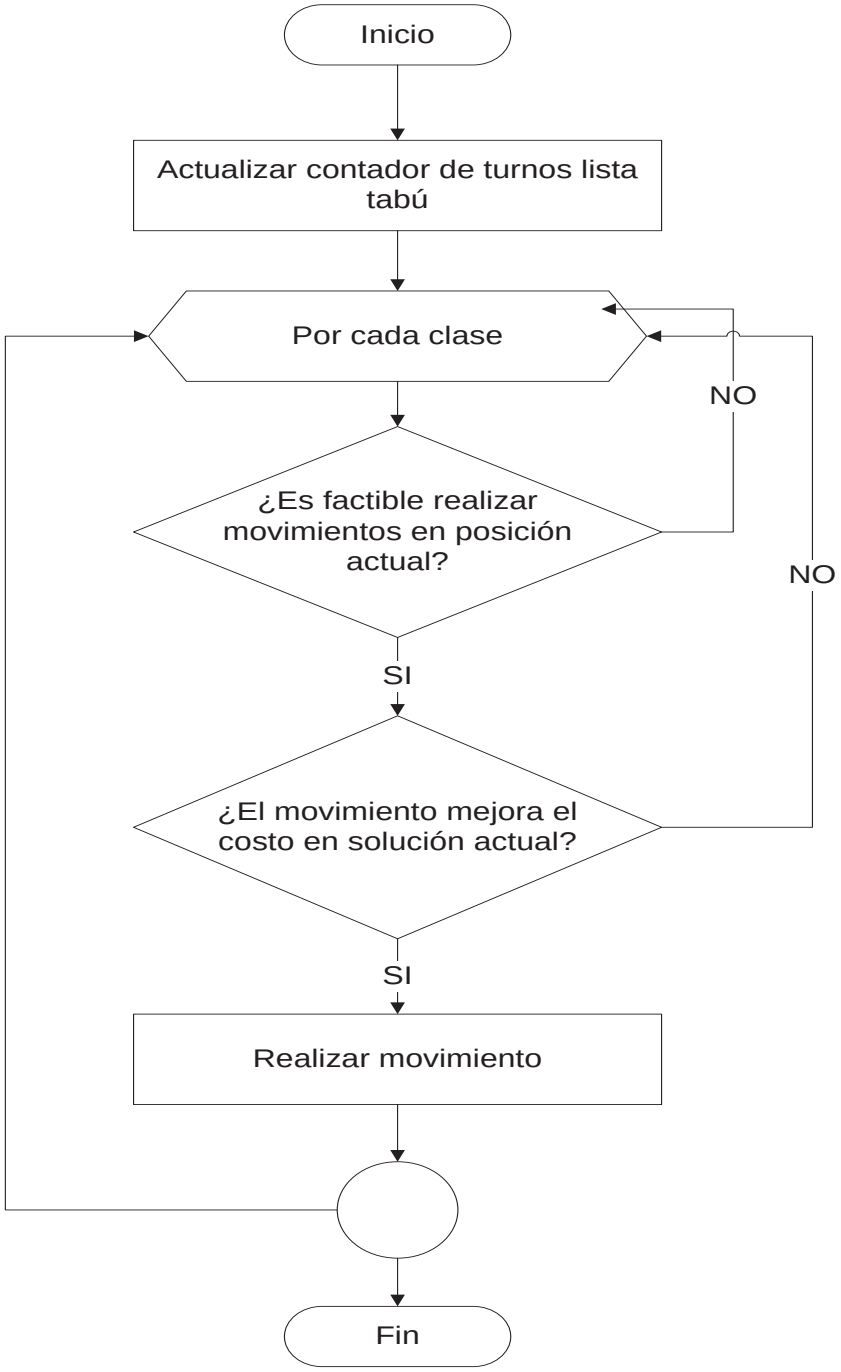


Figura 10 Intensificar

Intensificar se encarga de iterar sobre cada clase semanal de cada curso creando una lista de todos los movimientos factibles posibles, y que mejoren el costo actual. Luego de hacer esto se realizan uno a uno los movimientos en la lista en orden descendente de mejora de costos, reevaluando la factibilidad y el beneficio de costos a medida que la solución actual cambia. Si alguno de los movimientos se torna infactible o no contribuye a mejorar el costo global, simplemente no se realiza y se continúa con el siguiente movimiento candidato.

6.2 Implementación Propuesta del Modelo

Se ha elegido implementar el algoritmo en lenguaje c, por ser de un nivel más bajo comparado con los lenguajes orientados a objetos, lo que otorga, en general, una eficiencia mayor de ejecución.

La interfaz se implementará en Java, un lenguaje orientado a objetos y de alto nivel, el cual puede otorgar una menor complejidad de codificación y mayor dinamismo en el proceso de desarrollo.

Las estructuras de datos usadas para representar el modelo se muestran continuación y sus funciones serán descritas
a continuación.

```
Struct curso {
  int id;
 int cupo;
  int clases_semana;

};
```

Representa un curso para las asignaciones, incluyendo su cupo de alumnos y su número clases semanales.

```
Struct profesor {

  int id;
  int *claves_prioridad;
  int claves_prioridad_len;
  int *cursos_index;
  int cursos_index_len;
};
```

Representa a un profesor, junto con sus preferencias de horario y sus cursos asociados.

```
Struct sala {

  int id;
  int capacidad;

};
```

Representa una sala de clases para las asignaciones.

Matriz de Asignaciones

Fundamentalmente el algoritmo de asignación propuesto consiste en una matriz de punteros *struct_curso* los que en el estado inicial tendrán valor NULL, y al ser asignados apuntarán a un elemento de la lista de cursos.

Varias asignaciones pueden apuntar al mismo curso, ya que pueden hacerse varias clases a la semana de un mismo curso, la cantidad de veces que un curso se debe asignar está dada por la variable *n_clases_requeridas* en la estructura _curso, esta variable tiene el comportamiento de contador e irá disminuyendo a medida que se asignen las clases.
Las dimensiones de la matriz son:

Filas : cantidad de claves, convención de índice : i, convención del máximo: m

Columnas : cantidad de salas, convención de índice j, convención del máximo: n

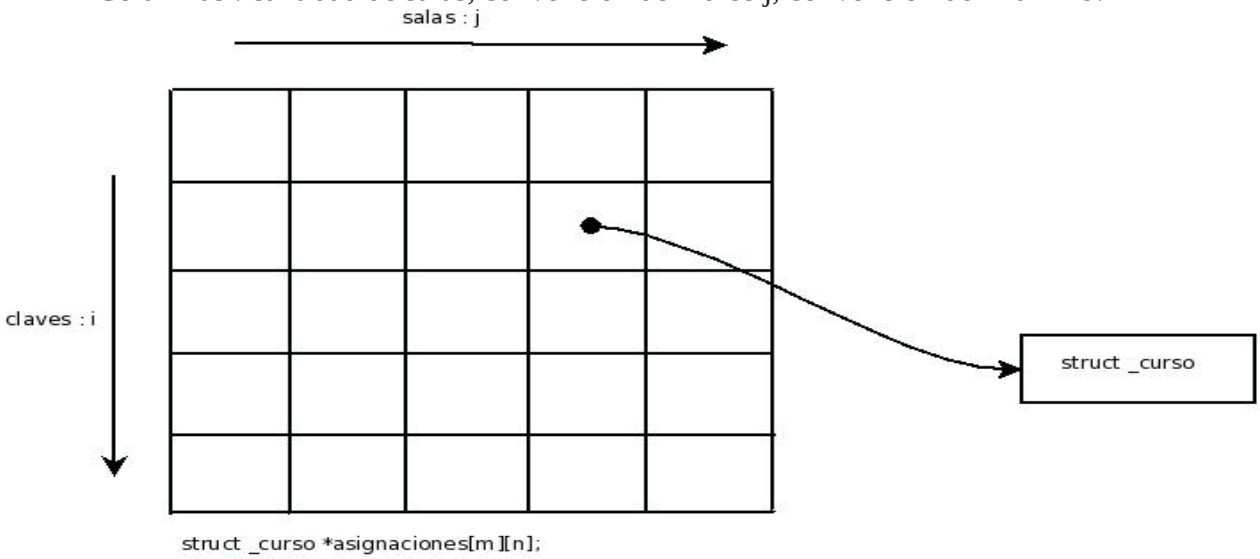


Figura 11 Matriz de Asignaciones

Disponibilidades de Horario de los Profesores

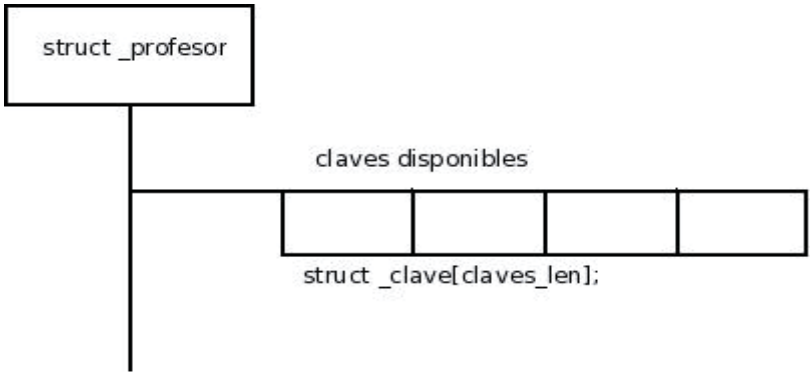


Figura 12 Disponibilidad de Horarios Profesores

Cada estructura de _profesor tendrá como propiedad un vector de estructuras _clave, que representarán todos los horarios disponibles para ese profesor en particular, el vector estará ordenado desde el horario con menor costo hasta el de mayor costo, para manejar los horarios con igual costo cada estructura _clave incluirá una propiedad (entero) que representa su costo.

Listado de cursos

Internamente, los cursos se guardarán en un vector, ya que el algoritmo está basado en índices, es la alternativa más lógica.

K es la convención de variable índice para los cursos, p es el número total de cursos

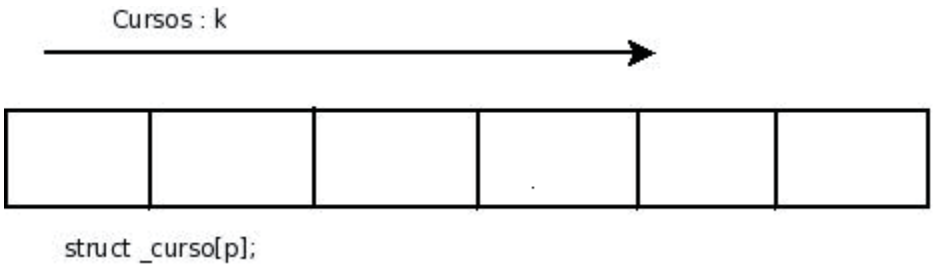


Figura 13 Listado de Cursos

Listado de Profesores

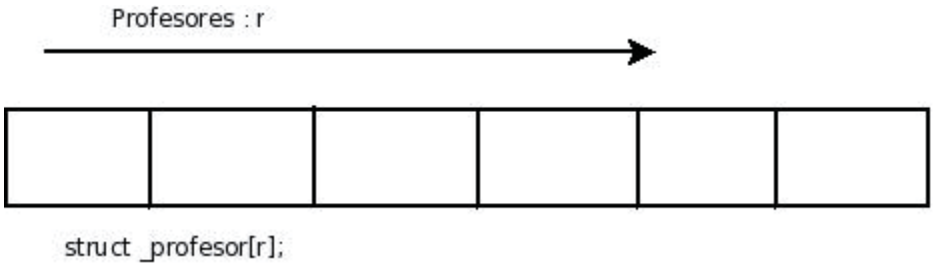


Figura 14 Listado de Profesores

Internamente los profesores están guardados en un vector de largo q (convención de máximo de profesores) de estructuras _profesor, y se usará r como convención para su índice.

Listado de Salas

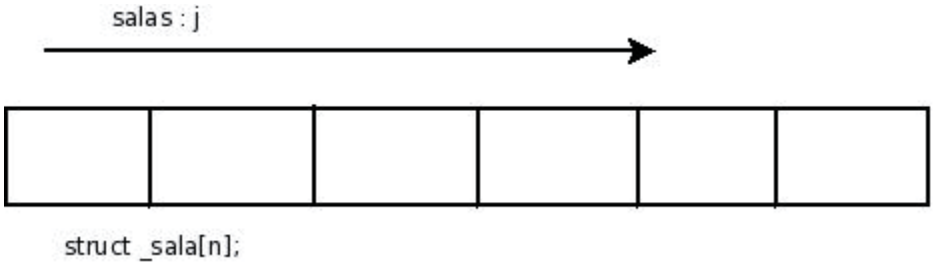


Figura 15 Listado Salas

Internamente las salas se guardarán en un vector de largo n (convención de largo) de estructuras _curso y se usará j como convención de índice.

Lista de Movimientos Tabú

Para el método de resolución aquí propuesto, la Búsqueda Tabú, se requiere una serie de datos que indiquen los movimientos que están bloqueados en cada iteración del algoritmo, conocida como lista tabú.

La lista tabú se ha implementado como una lista simplemente enlazada que contiene los identificadores y las posiciones de dos clases que fueron intercambiadas (flip), además del número de iteraciones que han permanecido en la lista (contador).

Antes del comienzo de cada iteración del algoritmo se recorre la lista tabú, aumentando en uno el contador de turnos de todos los elementos de la lista, y además eliminando los elementos que hayan cumplido con el número de turnos establecido. El número exacto de turnos que los elementos permanecen en la lista es un parámetro del algoritmo y se mantiene a lo largo de toda su ejecución.

Al hacer un movimiento en la fase de intensificación, intercambio o reasignación, siempre se verifica que no exista en la lista tabú, de lo contrario no se realiza.

Aplicación del criterio de aspiración

El objetivo de la lista tabú es no reversar movimientos realizados recientemente, los que contribuyen al estancamiento del algoritmo, pero pueden haber excepciones en las que la realización de un movimiento tabú puede mejorar la mejor solución encontrada, para que estos casos existe el criterio de aspiración hace una excepción en el criterio tabú para mejorar la solución global.

El criterio de aspiración está implementado haciendo una verificación de la diferencia incurrida en el costo global al hacer un movimiento durante la fase de

intensificación, si el nuevo costo global es menor que el mejor costo encontrado, el movimiento se realizará sin pasar por la verificación tabú.

Manejo de números pseudo-aleatorios

En la implementación del solver se utiliza directamente la función estándar de c *rand()*, la que provee una distribución suficientemente uniforme de números pseudo-aleatorios para los requerimientos del algoritmo.

La inicialización del generador de números pseudo-aleatorios se hace mediante la función estándar de c *srand()*, en la que se pasa como parámetro la salida de la función *time()* que devuelve la fecha actual, representada en el número de segundos desde el 01/01/1970. Con esto se espera un desempeño suficiente para las operaciones de selección e intercambio (pseudo)aleatorios.

6.3 Diagramas de Casos de Uso

Con los diagramas de casos de uso que se presentarán a continuación se pretende mostrar cómo el sistema implementa su interfaz gráfica y como se presentarán los datos obtenidos como solución al usuario.

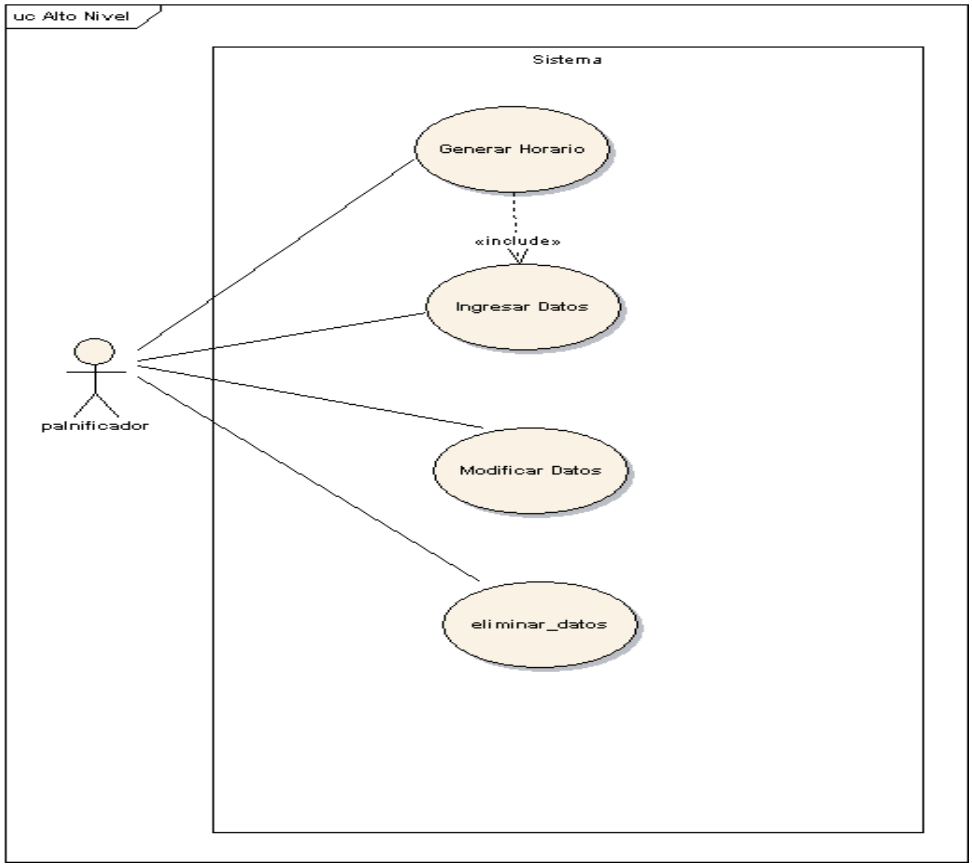


Figura 16 Diagrama de Caso de Uso Alto Nivel

Dentro de este diagrama de alto nivel se busca mostrar de forma general las acciones que el planificador puede realizar con el sistema. La función principal del sistema es generar los horarios de clases lo cual se muestra en la figura pero para esto necesita que con anterioridad se hayan ingresado los datos necesarios para realizar el cálculo. También se hace referencia en el diagrama a otras funcionalidades que son indispensables para que el sistema tenga un buen funcionamiento como son la posibilidad de modificar los datos que se han ingresado con anterioridad o eliminar los datos que ya no son necesarios.

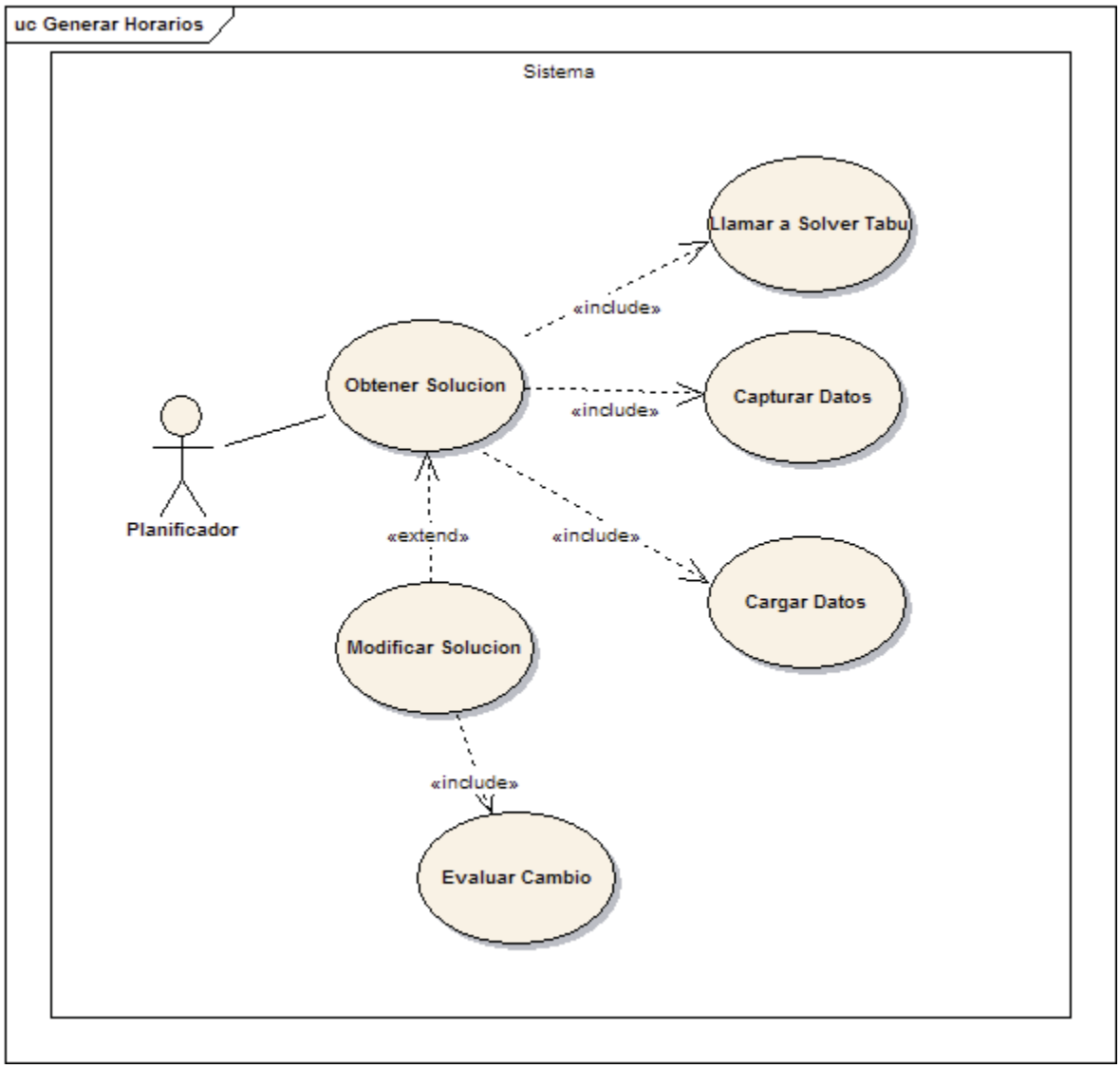


Figura 17 Diagrama de Casos de Uso Generar Horario

El caso de uso generar horario pretende ilustrar los pasos que se realizarán cuando el planificador pretenda obtener una solución para algún horario en un semestre, para lo cual se hace referencia a la llamada que se hace al modulo encargado de realizar los cálculos necesarios, la captura de los datos que devuelve dicho modulo y la carga de los datos para

se muestren al usuario, además ilustra que el usuario puede modificar la solución preliminar obtenida para lo cual el sistema evaluará si se ha violado alguna restricción del problema.

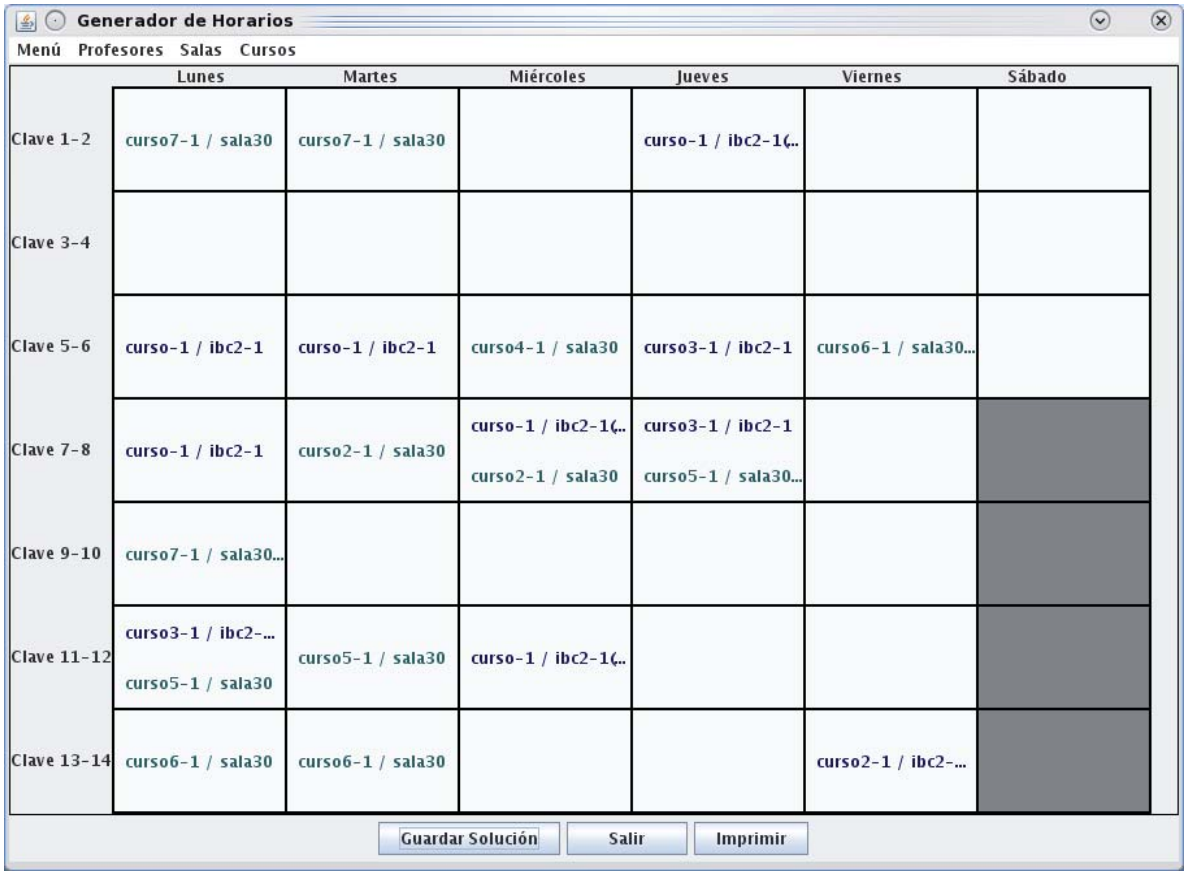


Figura 18 Interfaz Gráfica Para los Horarios

La figura anterior muestra una de las interfaces gráficas mediante la cual el sistema mostrará los horarios de clases entregados por el algoritmo de Búsqueda Tabú, ésta interfaz también es la pantalla principal del sistema desde la cual se puede acceder a las funcionalidades que se ofrecen. En cada clave horaria se mostrará al curso que tiene asignado más la sala de clases en la que se dictara el curso y en caso que sea una ayudantía también se especificará mediante la sigla (**AY)** al lado de la sigla o identificador de dicho curso.

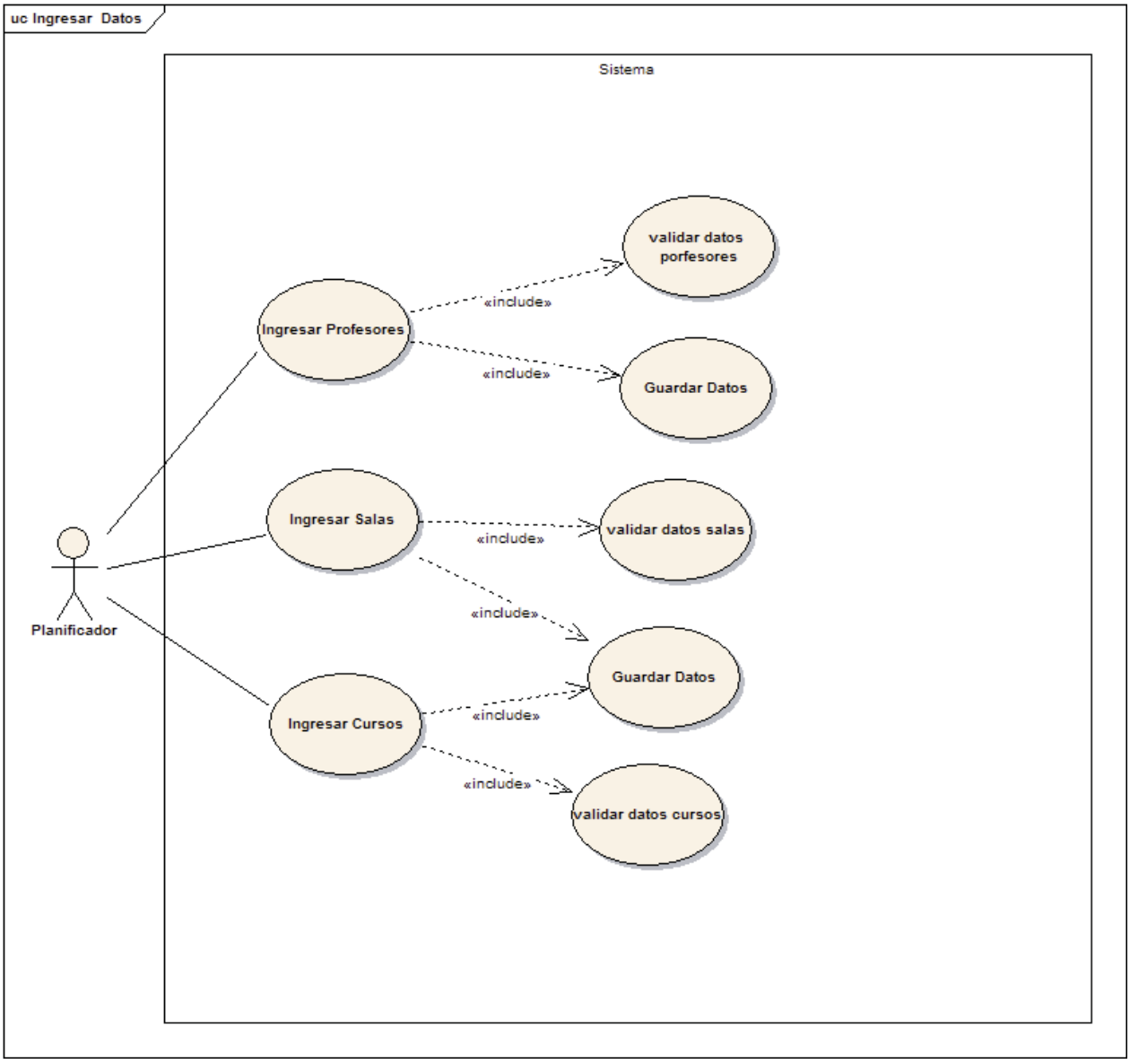


Figura 19 Diagrama de Caso de Uso Ingresar Datos

El caso de uso ingresar datos muestra los datos que el usuario debe ingresar para poder calcular una solución, estos datos deben ser validados para no se ingresen datos corruptos que puedan alterar el correcto funcionamiento del programa, luego posteriormente a la validación de los antecedentes estos se guardan para su uso posterior en los cálculos necesarios para lograr solucionar el problema.

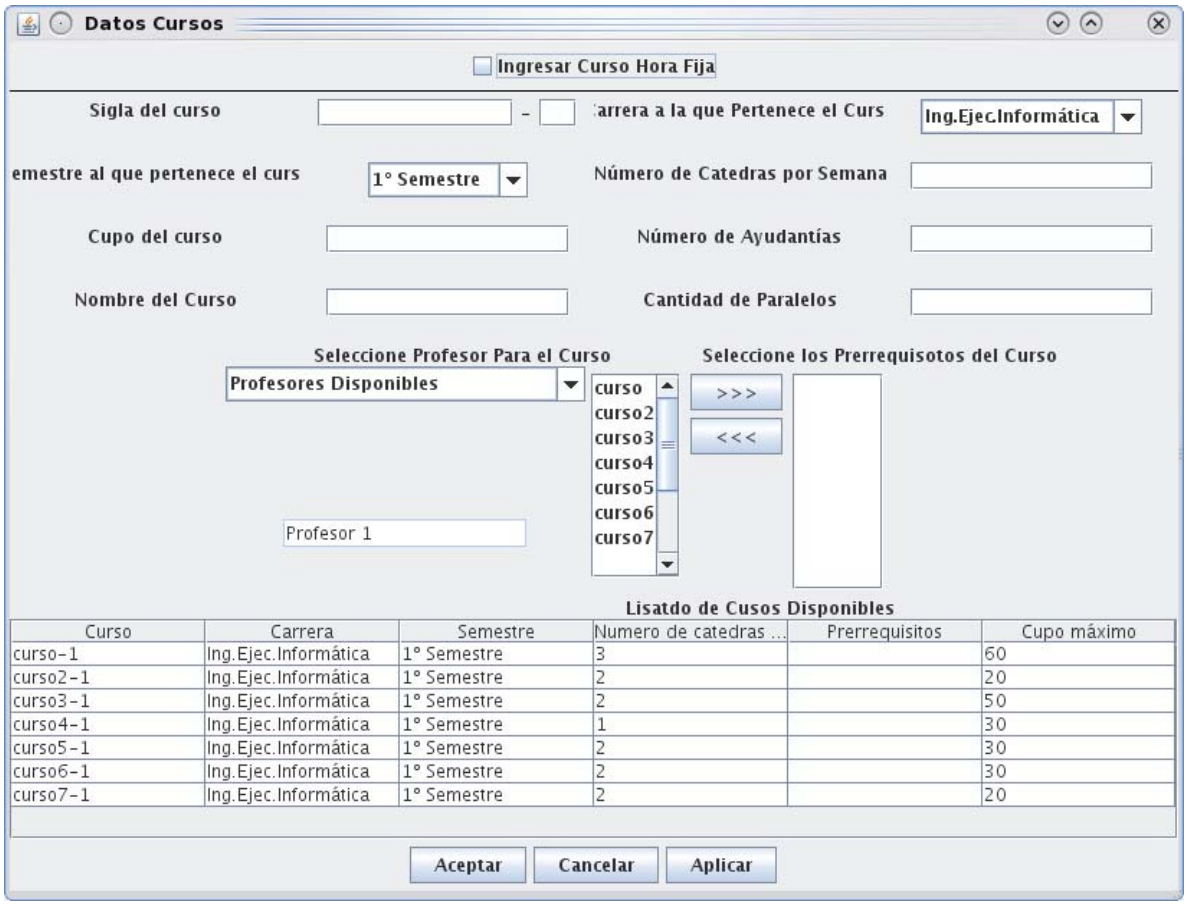


Figura 20 Interfaz Gráfica Ingresar Cursos

La figura 20 muestra la interfaz gráfica mediante la cual el usuario ingresará los datos de los cursos pertenecientes al semestre que se le desea obtener un horario de clases, para esto en la interfaz anterior se piden todos datos necesarios del curso incluyendo la carrera, semestre, ayudantías, prerrequisitos etc. Además se muestra una lista de los cursos que se han ingresados con anterioridad para tener claro que curos ya se encuentran en el sistema y no ingresarlos nuevamente. Los botones que se ven en la parte baja de la figura muestran las acciones que se pueden realizar, aplicar en caso que se deseen seguir ingresando cursos con lo cual se guardarán los datos del curso ingresado y se limpiara la pantalla para ingresar un curso nuevo, aceptar con lo cual se ingresara el curso y retornará a la pantalla principal guardando previamente los datos ingresados y por ultimo cancelar que su efecto es volver a la pantalla principal del programa sin realizar ninguna acción.

Figura 21 Interfaz Gráfica Ingresar Profesor

La figura 21 ingresar profesor se muestra los datos de los profesores que son el nombre y las preferencias horarias asociadas al catedrático que se está ingresando, con respecto a las preferencias horarias se deben ingresar en el orden que el profesor prefiere hacer clases de mayor a menor preferencia marcando cada uno de los checkbox en la forma anteriormente descrita, también se da la posibilidad de marcar un día completo al marcar el checkbox que está al lado del nombre de cada día de la semana, por último el nombre del profesor se ingresa en el espacio destinado para esto en la parte superior de la pantalla.

Figura 22 Interfaz Gráfica Ingresar Salas

La figura anterior muestra la forma de ingresar las salas y los datos necesarios de ésta, se debe ingresar sigla o código de la sala más la capacidad de alumnos que ella posee, en la lista de la parte baja se irán mostrando las salas ya ingresadas y su capacidad.

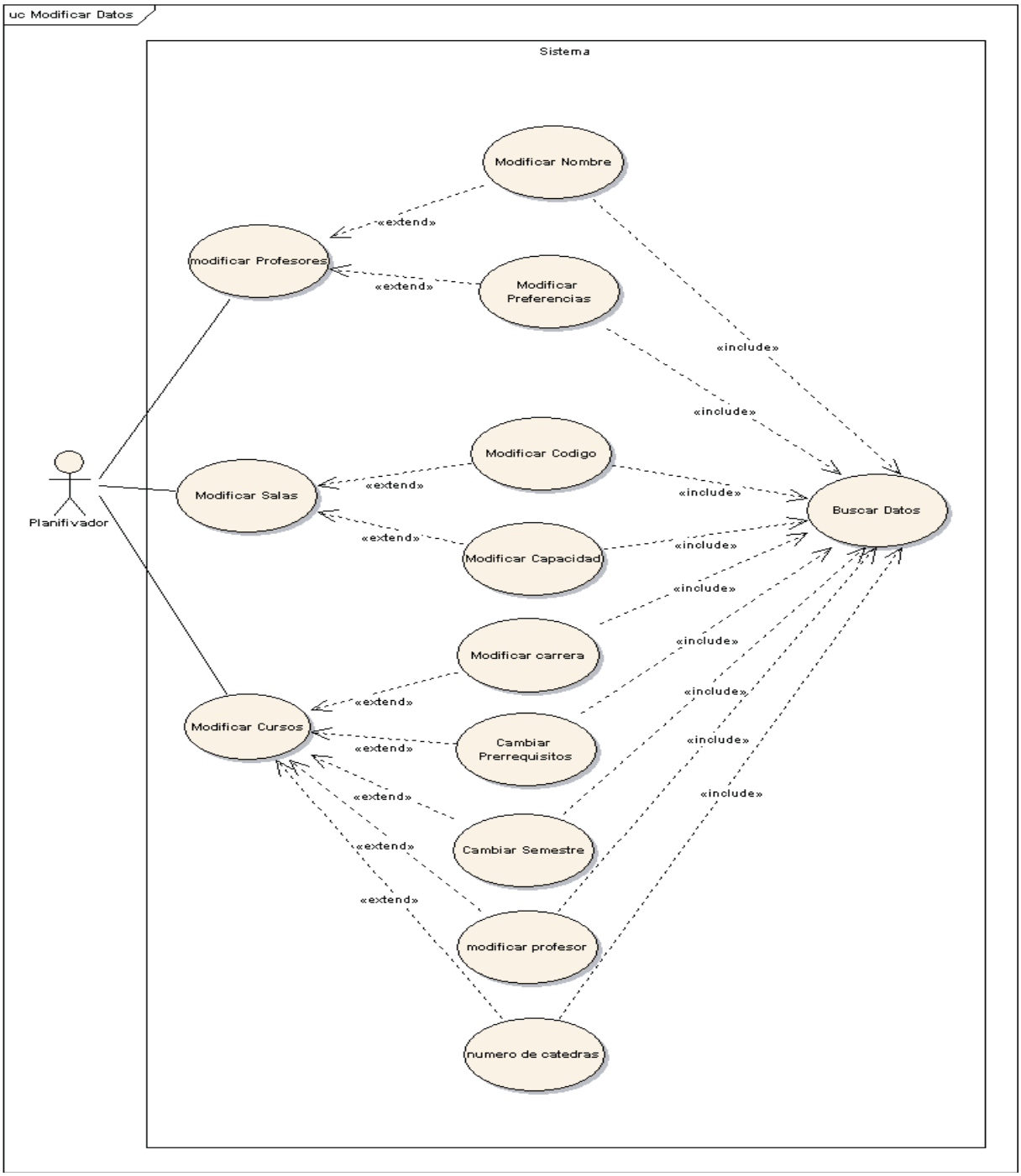


Figura 23 Diagrama de Caso de Uso Modificar Datos

El caso de uso modificar datos muestra las acciones que el sistema debe realizar en caso que un usuario quiera cambiar algún dato que se encuentre defectuoso o que haya

cambiado en el tiempo. Además muestra que para modificar los datos el sistema debe buscarlos para que el usuario pueda realizar la acción de actualizar lo que se crea necesario.

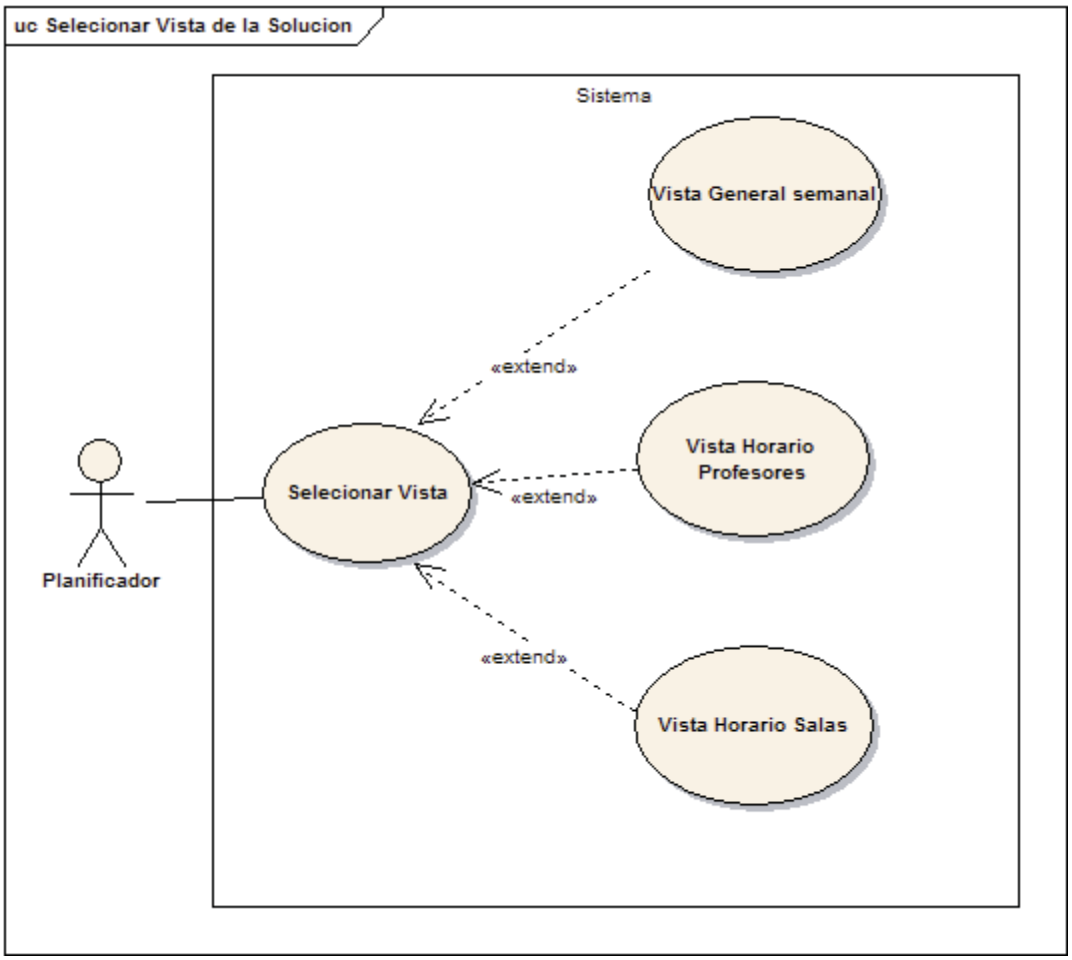


Figura 24 Diagrama de Caso de Uso Seleccionar Vista de la Solución

El caso de uso seleccionar vista de la solución pretende ilustrar que el usuario podrá tener más de una forma de visualizar el resultado obtenido. Estas vistas de la solución pueden ser en torno al horario de cada profesor de acuerdo a las clases que se le han asignado, a las salas para saber en qué horario se encuentran ocupadas o disponibles para su uso y la más importante que es la que se presenta en la pantalla principal del programa que es la vista de todos los cursos asignados y con sus respectivas salas.

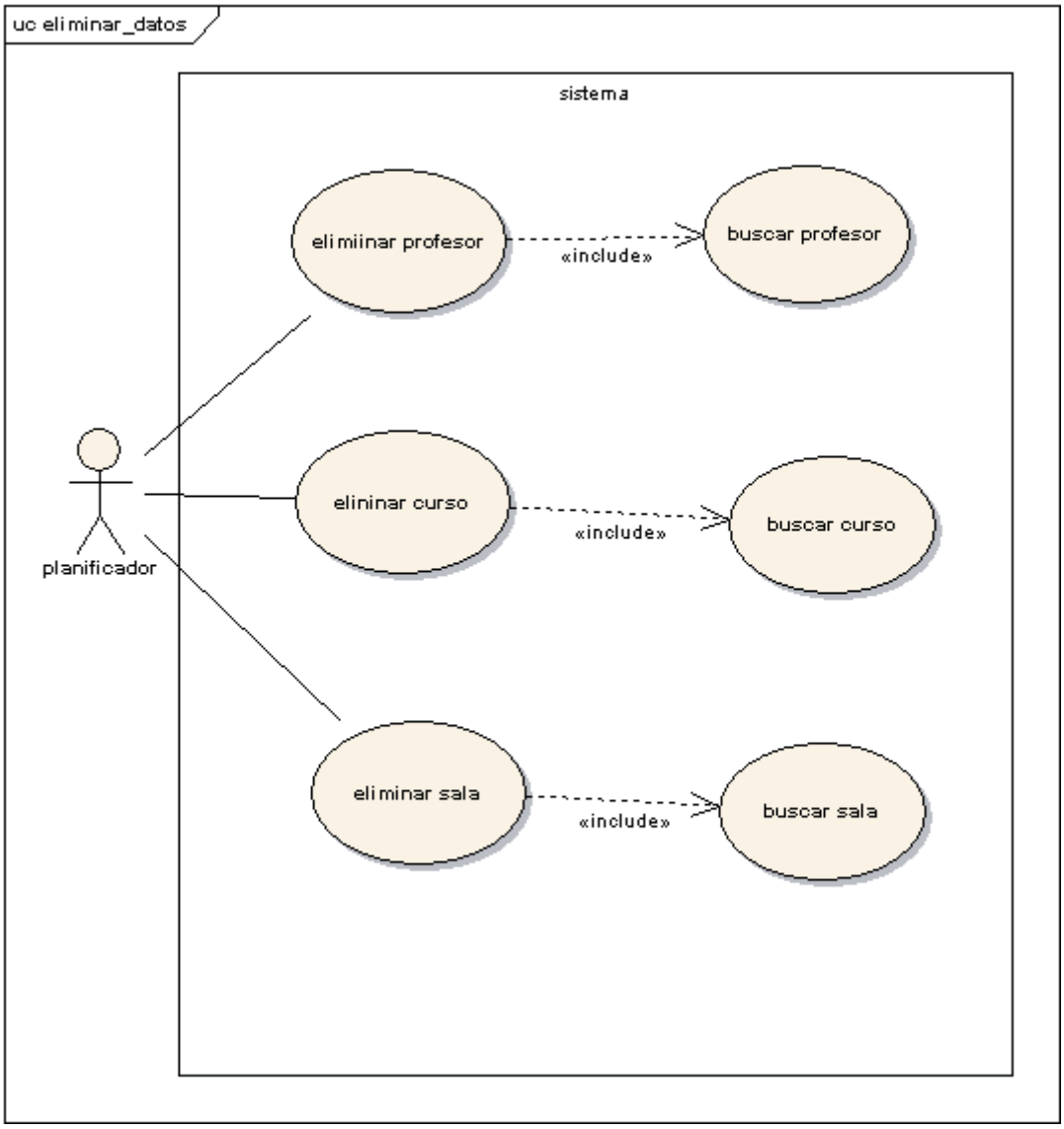


Figura 25 Diagrama de Casos de Uso Eliminar Datos.

El diagrama de casos de uso eliminar datos figura Nº25 muestra los datos que el sistema permitirá que sean eliminados y que el usuario podrá eliminar de acuerdo a lo que estime necesario, en cada uno de los casos el sistema buscara lo datos que se desean eliminar y posteriormente serán borrados del sistema.

6.4 Casos de Uso Narrativos

A continuación s expresan los principales caso de uso de forma narrativa extendida explicando paso a paso las acciones que puede tomar el planificador y la respuesta que el sistema entregara de acuerdo con lo que el planificador solicite.

Caso de Uso:	Generar Horarios.
Actor(es):	Planificador.
Propósito:	Generar un horario de clases válido para una semana de clases.
Tipo:	Principal y Esencial.
Referencias cruzadas:	Caso de uso ingresar datos.
Descripción:	El encargado de realizar la planificación semestral accede al sistema con el propósito de generar un horario de clases valido para el semestre que se cursara con la posibilidad de modificar la solución entregada si lo estima necesario.

Curso normal de eventos

Acción de los Actores	Respuesta del Sistema
1. el caso comienza cuando el encargado de la planificación necesita proyectar donde y cuando se dará cada clase correspondiente al semestre para ello el encargado accede al sistema en la opción generar horarios.	2. El sistema llama al programa encargado (solver Tabú) de generar el horario tentativo para el semestre de clases.
	3. El sistema rescata la solución que entrega por el solver.
	4. el sistema lee los datos obtenidos.
	5. el sistema interpreta los daros que ha entregado el solver.
	6. El sistema carga los datos de forma ordenada en la interfaz grafica.
7. El encargado revisa la solución y elige si: a) Aceptarla b) Modificarla	7. al sistema guarda la solución obtenida.
7.b. El encargado modifica la solución sugerida.	8. El sistema analiza si el cambio es posible.
	9. El sistema acepta y se realiza el cambio.
	10. El sistema rechaza el cambio e impide su ejecución.
El encargado vuelve al paso 7	

Caso de Uso:	Ingresar Datos.
Actor(es):	Planificador
Propósito:	Ingresar los datos necesarios para generar los horarios de clases.

Tipo:	Principal y Esencial.
Referencias Cruzadas:	No.
Descripción :	El encargado de realizar la planificación semestral debe ingresar los datos necesarios para generar el horario de clases tentativo para el semestre.

Curso normal de los eventos

Acción de los actores	Respuesta del sistema
1. El encargado de asignar horarios entra a ingresar datos	2. El sistema entrega las opciones a) datos de profesores b) datos de salas c) datos de cursos
3. El encargado selecciona una opción a- Opción a) b- Opción b) c- Opción c)	4. el sistema despliega la opción seleccionada
	3.a El sistema pide los datos de los profesores ,sus preferencias de horarios y los almacenan
	3.b El sistema pide las salas, las capacidades de cada una de ellas y los almacena
	3.c El sistema pide los cursos que se dictan en el semestre, el semestre al que pertenece el curso, la carrera del curso, los prerrequisitos del curso y los almacena
5. Salir de ingresar datos	6. El sistema vuelve a la pantalla principal

Caso de Uso:	Modificar datos.
Actor(es):	Planificador
Propósito:	Modificar datos necesarios para el sistema que se habían ingresado con anterioridad.
Tipo:	Principal.
Referencias Cruzadas:	Caso de uso ingresar datos.
Descripción:	El encargado de realizar la planificación necesita ingresar al sistema para corregir datos que se ingresaron mal o que han cambiado desde que se ingresaron.

Curso normal de eventos

Acción de los actores	Respuesta del sistema

1. El encargado de los horarios ingresa a modificar datos.	2. El sistema despliega las opciones a) modificar datos profesor b) modificar datos salas c) modificar datos cursos
3. El encargado selecciona una opción a- opción a) b- opción b) c- opción c)	4. El sistema despliega la opción seleccionada.
	3.a El sistema muestra los datos de los profesores para que puedan ser modificados y guarda las modificaciones.
	3.b el sistema muestra los datos de las salas y guarda las modificaciones.
	3.c el sistema muestra los datos de cursos para que sean modificados y luego guarda las modificaciones.
Salir de modificar datos	El sistema vuelve a la pantalla principal.

Caso de Uso:	Seleccionar vista de la Solución.
Actor(es):	Planificador.
Propósito:	Seleccionar una de las vistas disponibles para la solución del problema de asignación de salas.
Tipo:	Principal.
Referencias Cruzadas	Caso de uso generar horario
Descripción:	El planificador ingresa al sistema y selecciona la vista de la solución más apropiada para sus propósitos

Curso Normal de los Eventos	
Acción de los Actores	**Respuesta del Sistema**
1) El planificador ingresa al sistema	2) El sistema determina una solución para el problema y muestra la solución
	3) El sistema muestra las opciones para ver la solución. a)Vista general de la semana b)Vista de un día a una hora c)Vista de un día de la semana
4) El usuario selecciona una opción a- opción a)	5) El sistema muestra la opción seleccionada.

b- opción b) c- opción c)	
	4.a . El sistema muestra los datos de la solución que corresponden a una vista general de la semana donde en cada uno de los horarios se muestran todas las clases y salas asignadas en ese horario.
	4.b El sistema muestra los datos de la solución que corresponden a un día/hora específico.
	4.c El sistema muestra la solución detonas las clases y salsa asignadas a un día de la semana
El usuario selecciona salir	El sistema se cierra

6.5 Diagramas de secuencia

Los diagramas de secuencia que se muestran a continuación pretenden mostrar cuales son los pasos que se debe seguir en un orden temporal establecido encada uno de ellos de forma individual para lograr realizar una acción en el sistema

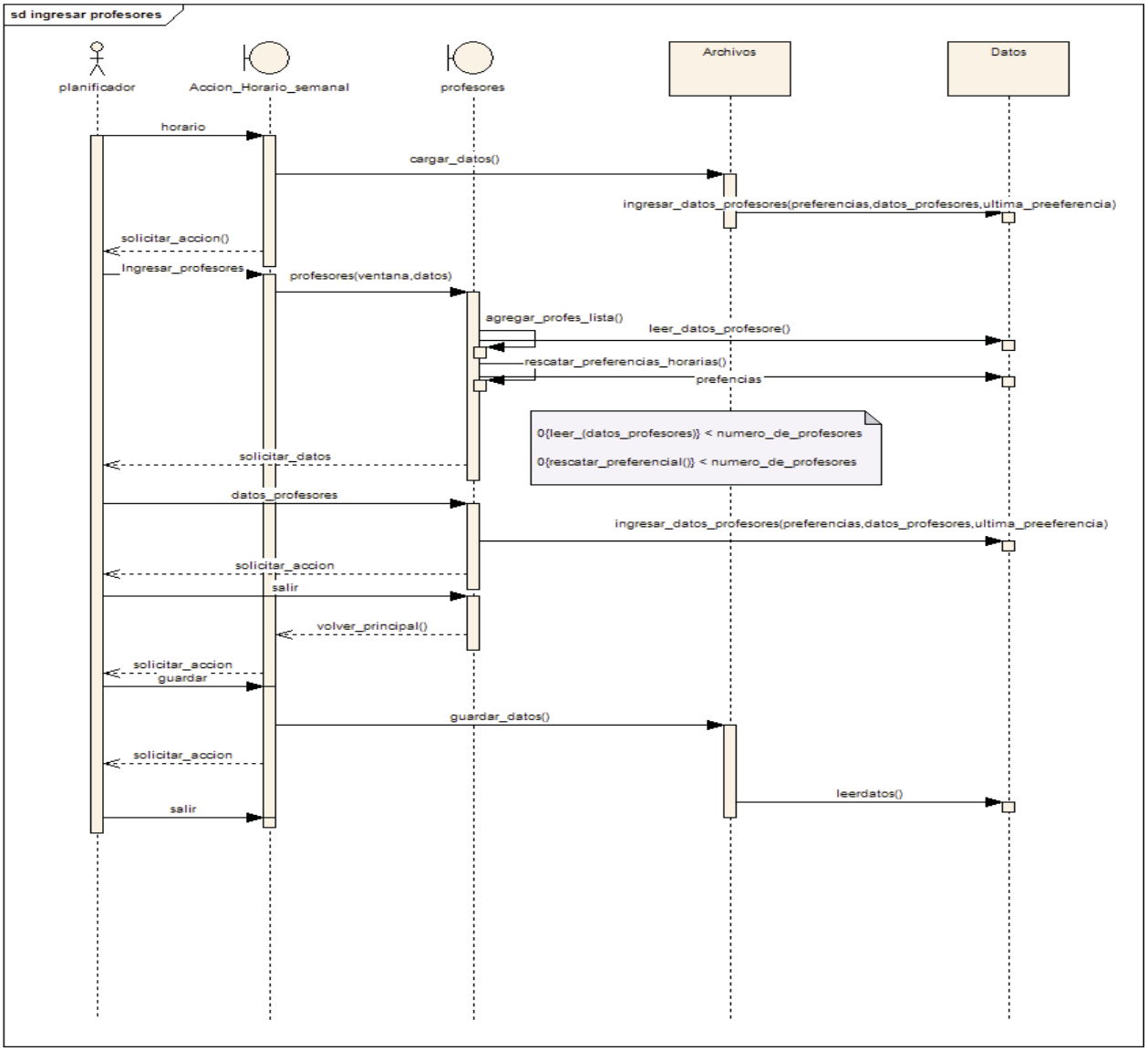


Figura 26 Diagrama de Secuencia Ingresar Profesores

El diagrama anterior que *obedece a uno de los escenarios del caso de uso ingresar datos ilustrado en la página número 55 de éste texto,* muestra la secuencia desde que el usuario entra al sistema, luego el sistema carga los datos de los profesores que ya se han ingresado con anterioridad después el usurario selecciona ingresar profesores el sistema abre al ventana destinada para este propósito y muestra en una tabla los datos que se han

ingresado con anterioridad, el usuario ingresa los datos del nuevo profesor presiona aceptar los datos se ingresan al sistema luego de esto cuando ya no quiera ingresar más profesores selecciona salir y vuelve a la pantalla principal donde debe seleccionar guardar para que los datos se almacenen en forma definitiva en el sistema.

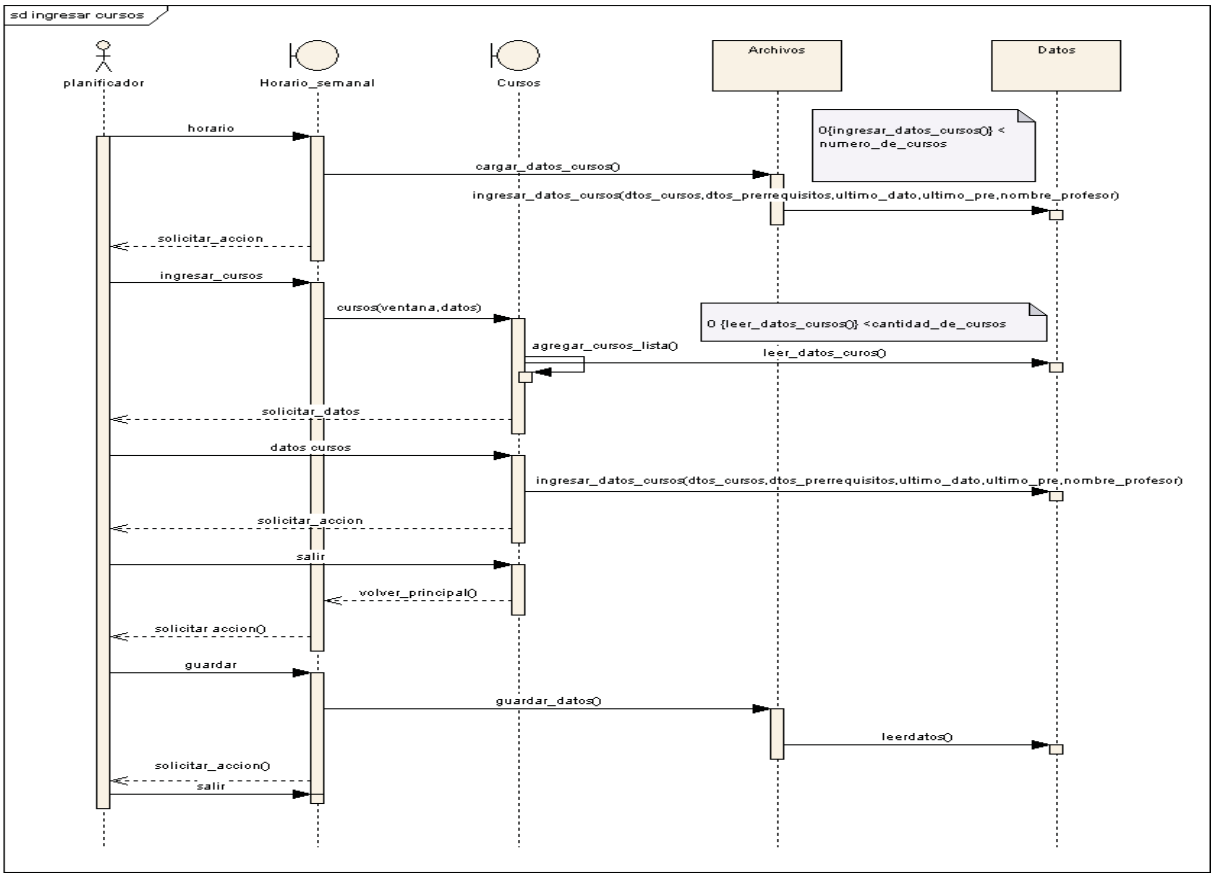


Figura 27 Diagrama de Secuencia Ingresar Cursos

El diagrama anterior *que concuerda con el caso de uso expuesto en la página 55 del documento e ilustra uno de los escenarios que en el diagrama ingresar datos se exponen*, muestra la secuencia desde que el usuario entra al sistema, luego el sistema carga los datos de los cursos que ya se han ingresado con anterioridad después el usurario selecciona ingresar curso el sistema abre la ventana destinada para este propósito y muestra en una tabla los datos de los cursos que se han ingresado con anterioridad, el usuario ingresa los datos del nuevo curso, presiona aceptar para que los datos se ingresen al sistema luego de esto cuando ya no quiera ingresar más cursos selecciona salir y vuelve a la pantalla principal donde debe seleccionar guardar para que los datos se almacenen en forma definitiva en el sistema.

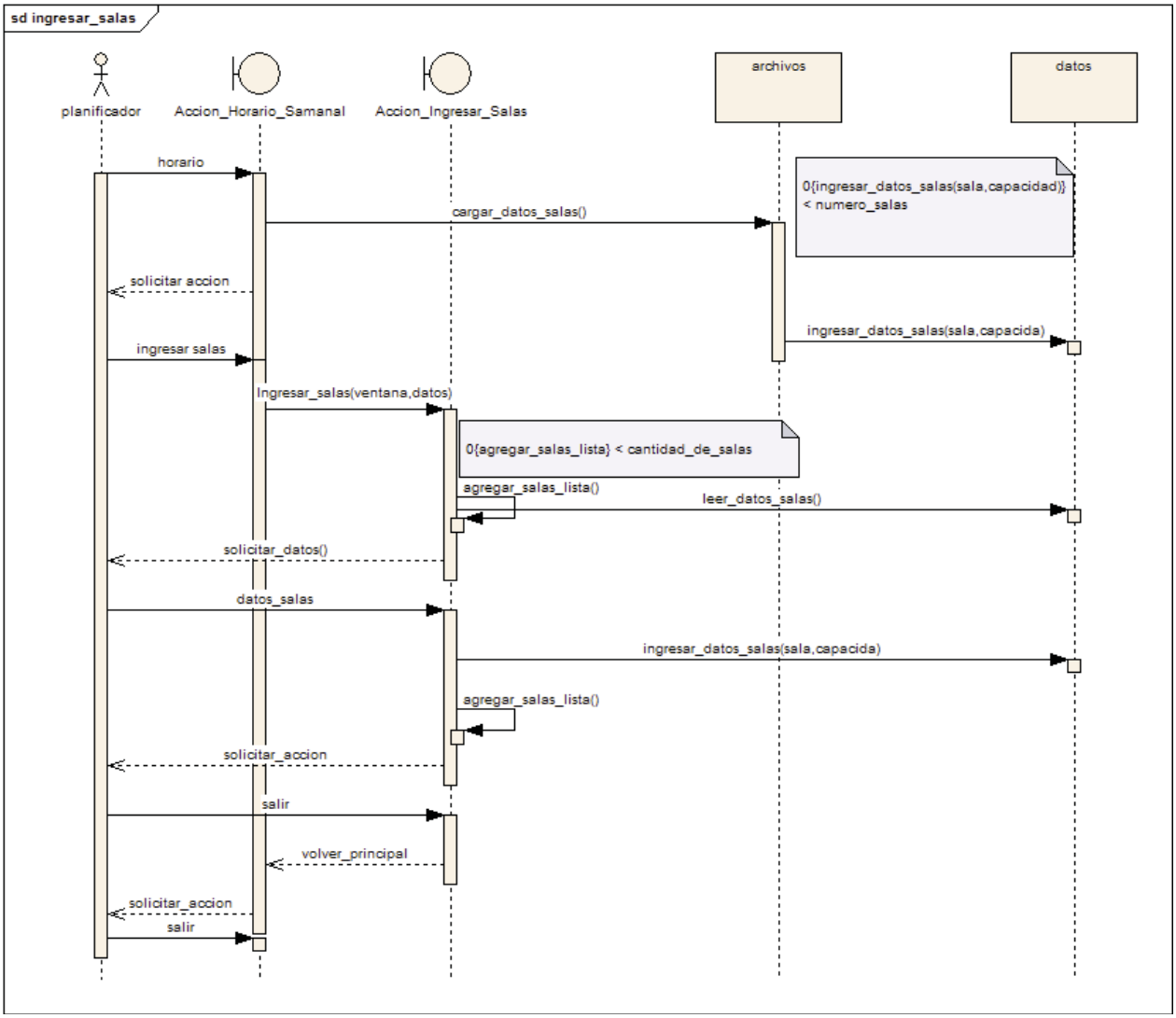


Figura 28 Diagrama de Secuencia Ingresar Salas

El diagrama anterior *se apega al escenario expuesto en el caso de uso ingresar datos en particular al escenario ingresar datos salas,* muestra la secuencia desde que el usuario entra al sistema, luego el sistema carga los datos de las salas que ya se han ingresado con anterioridad después el usurario selecciona ingresar salas el sistema abre la ventana destinada para este propósito y muestra en una tabla los datos que se han ingresado con anterioridad, el usuario ingresa los datos de la nueva sala (código de la sala, capacidad) presiona aceptar y los datos se ingresan al sistema luego de esto cuando ya no deseen ingresar más salas se selecciona salir y vuelve a la pantalla principal donde debe seleccionar guardar para que los datos se almacenen en forma definitiva en el sistema.

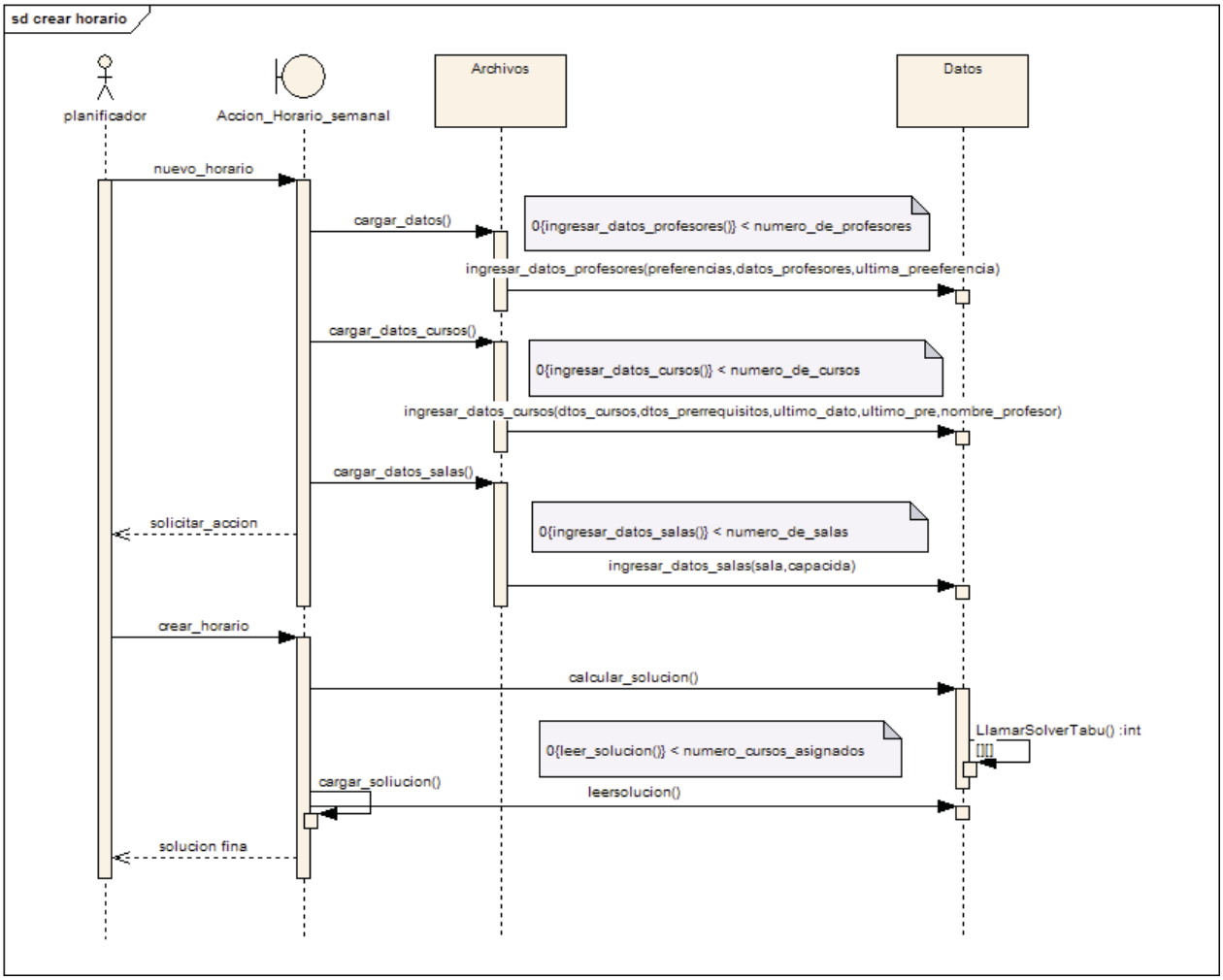


Figura 29 Diagrama de Secuencia Crear Horario

La figura anterior *concuerda con el diagramas de caso de uso expuesto en la figura Nº 17 página 53 generar horarios*, muestra la interacción y la secuencia desde que el usuario entra al sistema y se abre la pantalla principal donde verá la solución al problema. El sistema carga en memoria los datos que son necesarios para generar el horario, a continuación el usuario selecciona crear horario con lo cual el sistema llama al módulo encargado de realizar el cálculo, luego éste interpreta los datos y muestra la solución final en la pantalla.

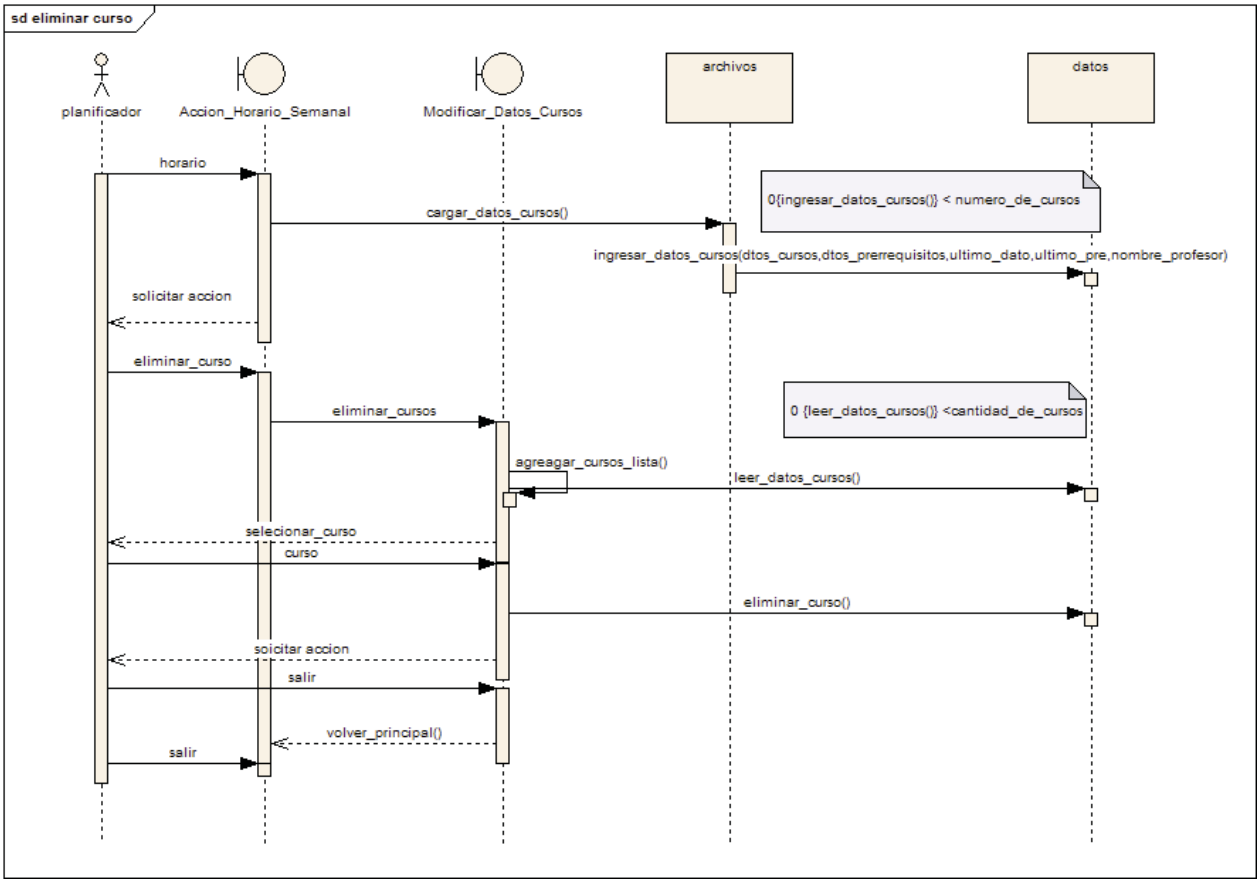


Figura 30 Diagrama de Secuencia Eliminar Curso

La figura Nº 30 que *gráfica uno de los escenarios del diagrama de caso de uso eliminar datos expuesto en la página 60 del documento*, muestra la secuencia de pasos desde que el usuario entra al sistema el cual inmediatamente carga los datos necesarios de los cursos que se han ingresado con anterioridad el usuario selecciona eliminar curso, con lo cual se crea la interfaz necesaria para este fin, los cursos que se encuentran en el sistema se cargan en una lista en donde el usuario podrá seleccionar el cuso que desea eliminar ,el sistema elimina el curso luego el usuario selecciona salir, vuelve a la pantalla principal en donde selecciona guardar para que los datos cambien de forma definitiva.

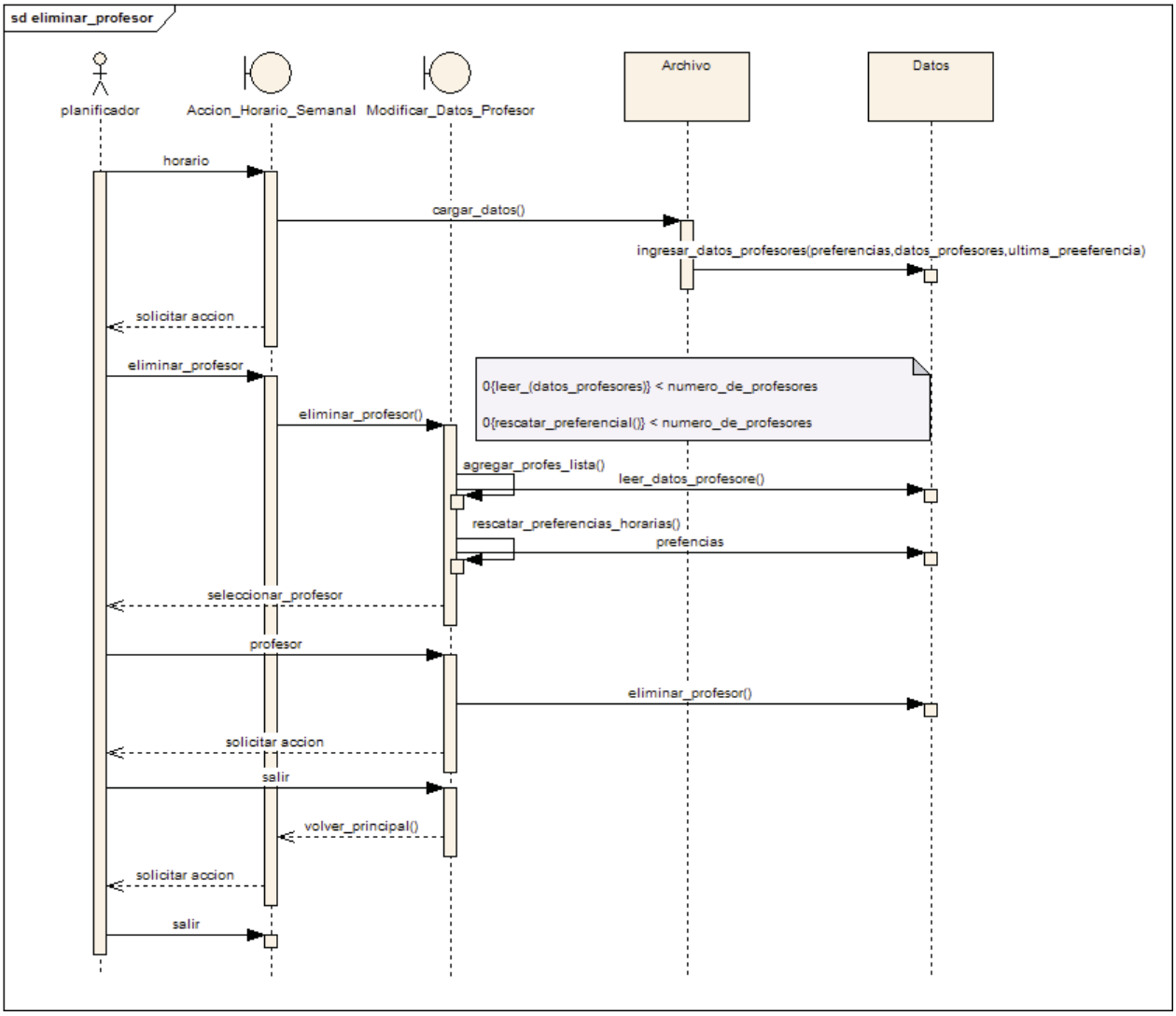


Figura 31 Diagrama de Secuencia para Eliminar Profesor

La figura Nº 31 que *gráfica uno de los escenarios del diagrama de caso de uso eliminar datos expuesto en la página 60 del documento,* muestra la secuencia de pasos desde que el usuario entra al sistema el cual inmediatamente carga los datos necesarios de los profesores con sus preferencias horarias, que se han ingresado con anterioridad el usuario selecciona eliminar profesor con lo cual se crea la interfaz necesaria para eliminar el dato, los profesores que se encuentran en el sistema se cargan en una lista en donde el usuario podrá seleccionar al profesor que desea eliminar ,el sistema elimina el o los datos del docente seleccionado. Luego el usuario selecciona salir, vuelve a la pantalla principal en donde selecciona guardar para que los datos cambien de forma definitiva.

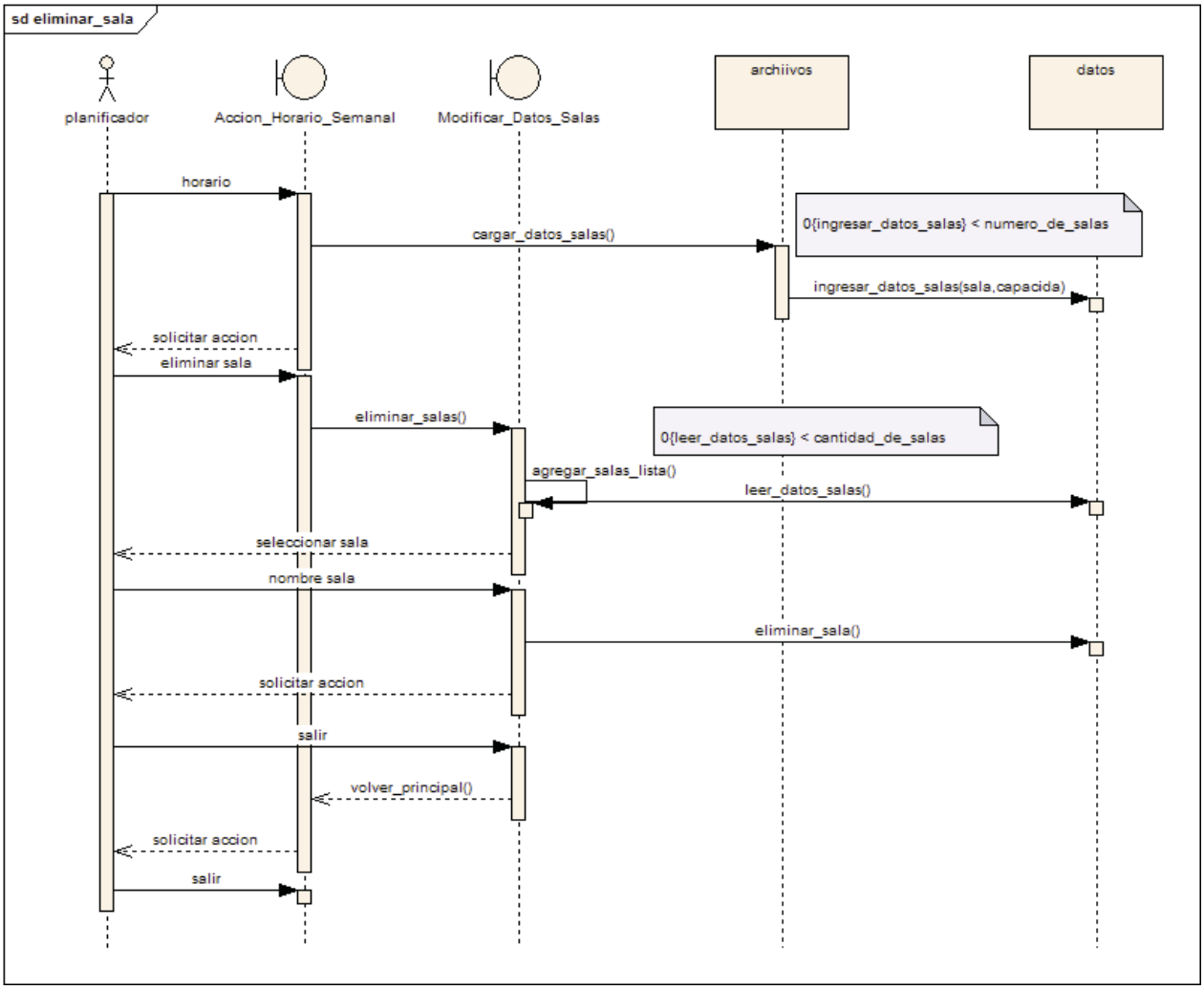


Figura 32 Diagrama de Secuencia para Eliminar Salas

La figura Nº 32 que *gráfica uno de los escenarios del diagrama de caso de uso eliminar datos expuesto en la página 60 del documento*, muestra la secuencia de pasos desde que el usuario entra al sistema el cual inmediatamente carga los datos necesarios de las salas, que se han ingresado con anterioridad, el usuario selecciona eliminar sala con lo cual se crea la interfaz necesaria para eliminar el dato, las salas que se encuentran en el sistema se cargan en una lista en donde el usuario podrá seleccionar la sala que desea eliminar, el sistema elimina el o los datos seleccionados. Luego el usuario selecciona salir, vuelve a la pantalla principal en donde selecciona guardar para que los datos cambien de forma definitiva.

6.6 Comunicación Entre las Distintas Tecnologías

Debido a que el sistema no se desarrolla en un solo lenguaje de programación sino que se hace en dos, uno orientado a objetos y de alto nivel, en particular JAVA que es el

lenguaje mediante el cual se implementa la interfaz gráfica para comunicarse con el usuario y un lenguaje estructurado en particular C que es el lenguaje mediante el cual el software realiza los cálculos y las acciones necesarias para lograr una solución de alta calidad es que se exponen los distintos tipos de diagramas los casos de uso y diagramas de secuencia para la parte orientada a objetos y los diagramas de flujo y de estructuras para la parte del software estructurada.

Debido a lo expuesto en el párrafo anterior se optó por presentar la lógica del solver como una librería con una interfaz bien definida para ingresar datos y para la entrega de la solución.

De esta manera, cualquier programa implementado en c, c++ o virtualmente cualquier otro lenguaje, mediante wrappers, puede hacer uso de la lógica del solver.

En este caso particular debió buscar una forma para que ambos lenguajes (C, JAVA) interactúen de manera fácil y fluida en esa búsqueda se encontró Java Native Interface (JNI) que es la manera mediante la cual este software comunicara ambos lenguajes.

6.6.1 Java Native Interface (JNI)

JNI es mecanismo que permite ejecutar código nativo desde java y viceversa.
El código nativo son funciones escritas en lenguaje de programación C o C++ para el sistema operativo donde se está ejecutando la maquina virtual de Java. Por maquina virtual de java se entiende al entorno de programación formado por.

- El entorno de ejecución de bytecode Java.
- El conjunto de clases de Java.

JNI posee una interfaz bidireccional que permite a las aplicaciones Java llamar código nativo y viceversa.

- **Native Methods.:** Que permite a Java llamar a funciones implementadas en librerías nativas.

- **Invocation Interface:** Que permite incrustar una maquina virtual java en una aplicación nativa

Existen dos formas principales en que JNI comunica el código Java con el código nativo.

Librerías de enlace estático: son ficheros destinados a almacenar funciones, clases y variables globales que tradicionalmente se crean a partir de varios ficheros de código objeto **.o** (UNIX) o **.obj** (WINDOWS). En Unix las librerías de enlace estático pueden tener la extensión **.a** y en Windows la extensión **.lib** [14].

Las funciones de la librería se incluyen dentro del ejecutable durante la fase de enlazado, con lo que una vez generado el ejecutable ya no es necesario disponer de las librerías de enlace estático.

Librerías de enlace dinámico: son ficheros cuyas funciones no se incrustan en el ejecutable durante la fase de enlazado, sino que en tiempo de ejecución el programa busca el fichero, carga su contenido en memoria y enlaza su contenido según va siendo necesario, es decir según vamos llamando a cada una de las funciones. Esto tiene la ventaja que varios programas pueden usar la librería al mismo tiempo. Lo cual reduce el uso de disco duro. Especialmente con las llamadas al sistema operativo que suelen ser usadas por muchas aplicaciones a la vez [14]. Su extensión también varía del sistema operativo que se esté usando.

Sistema Operativo	**Extensión**
Windows	.dll
Unix	.so

Tabla 7 Extensiones Para Sistemas Operativos

Para que los programas encuentren las librerías de enlace dinámico están deben ponerse en determinados directorios. Cada sistema operativo tiene sus propias reglas.

Sistema Operativo	**Reglas de búsqueda de librerías de enlace dinámico**
Windows	Busca los directorios donde está la variable (.exe) y en los directorios indicados por la variable de entorno PATH
Unix	Busca los directorios indicados en la variable de entorno LD_LIBRARY_PATH

Tabla 8 Reglas de los Sistemas Operativos para Buscar Librerías de Enlace Dinámico

CAPÍTULO 7:
Plan de Pruebas

El objetivo de este plan es probar las funcionalidades del sistema, por completo verificando todos los detalles que se puedan presentar, por lo que el objetivo de estas pruebas es medir el correcto funcionamiento de cada uno de los módulos del sistema en forma independiente y como estos módulos interactúan entre sí.

También el plan de pruebas nos permitirá saber cómo es usuario final se adapta al funcionamiento del software permitirá saber a ciencia cierta que tan usable es el sistema. Además existen las pruebas que el cliente (La Escuela de Ingeniería Informática) realizara una vez que el sistema se encuentre instalado y funcionando.

Para llevar a cabo el plan de pruebas es necesario:

1. Planificar las pruebas para cada modulo.
2. Diseñar casos de prueba que especifiquen que modulo probar y cómo hacerlo.
3. Revisar el resultado de las pruebas realizadas para probar el real funcionamiento del sistema.

Enfoque de Prueba

- **Caja Negra:** Se concentra sobre la interfaz del software (se comprueban las funciones específicas del sistema)
- **Caja Blanca:** Se basa en el minucioso examen de los detalles procedimentales (se comprueban los caminos lógicos del software)

Tipo de Prueba: Éste describe cómo se prueban los distintos componentes del sistema.

- **Caso de Prueba**: Específica la forma en que se probará el sistema, indica cómo debe ser la entrada y los resultados con los que se va a realizar la prueba. Entre los casos de prueba se puede mencionar:
- **Pruebas Exhaustivas:** Se basa en poner al sistema en todas situaciones posibles para encontrar hasta la última falla.

Procedimiento de Prueba: Indica cómo hacer la realización de uno o varios casos de prueba.

Plan de Prueba: Permite describir estrategias, recursos y planificación de la prueba. Se definen especificaciones del tipo de prueba que se va a utilizar y qué objetivos debe lograr.

Planificación de Prueba: Se determinará qué pruebas se van a realizar, cuándo y a qué partes del sistema.

Evaluación de Prueba: Se evaluará los resultados que arrojaron las pruebas realizadas al sistema

7.1 Enfoques de Prueba

A continuación se hará referencia a los distintos enfoques de pruebas que se le pueden realizar al software para asegurar la calidad y su funcionamiento óptimo.

7.1.1 Pruebas exhaustivas

La prueba ideal de un sistema sería exponerlo en todas las situaciones posibles, así encontraríamos hasta el último fallo. Indirectamente, garantizamos su respuesta ante cualquier caso que se le presente en la ejecución real.

Esto es imposible desde todos los puntos de vista: humano, económico.

Dado que todo es finito en programación (el número de líneas de código, el número de variables, el número de valores en un tipo, etc.) cabe pensar que el número de pruebas posibles es finito. Esto deja de ser cierto en cuanto entran en juego bucles, en los que es fácil introducir condiciones para un funcionamiento sin fin. Aún en el irrealista caso de que el número de posibilidades fuera finito, el número de combinaciones posibles es tan enorme que se hace imposible su identificación y ejecución a todos los efectos prácticos.

Sobre esta premisa de imposibilidad de alcanzar la perfección, hay que buscar formas humanamente abordables y económicamente aceptables de encontrar errores.

7.1.2 Pruebas de Caja Blanca

También son conocidas como pruebas de estructurales. En este tipo de prueba estamos siempre observando el código y se usan las estructuras de control del diseño para obtener los casos de prueba, a través estos el ingeniero de software podrá probar que todos los caminos lógicos de cada modulo se ejecuten por lo menos una vez. Entre las pruebas de caja blanca se encuentran:

Pruebas de Segmentos: este tipo de pruebas se enfoca en las partes del código que no tiene puntos de decisión, el computador está obligado a ejecutarlas una tras otra por lo que se puede decir que se han probado todos los segmentos o sentencias .Aunque en estricto rigor el proceso de pruebas termina antes de ejecutar el total de los segmentos ya que resultaría demasiado costoso y necesitaría demasiado tiempo para ejecutar todas y cada una de las sentencias.

Pruebas de Ramas: Estos casos son un refinamiento de las pruebas de segmentos ya que consiste en recorrer todas las posibles salidas de los puntos de decisión (if else) y basta con ejecutar una vez con éxito la condición para probarlas.

Pruebas de Ciclos o Bucles: Los ciclos no son más que segmentos controlado por decisiones por lo que en teoría las pruebas de ramas bastarían, pero esto es solo en teoría ya que los ciclos podrían ser una fuente inagotable de errores. Ya que un ciclo se ejecuta un cierto número de veces; pero ese número debe ser muy preciso, y basta con que se ejecute una vez más o menos para que esto tenga consecuencias indeseables.los ciclos while deben probarse con cero ejecuciones una ejecución y más de una ejecución en cambio un ciclo for basta con ejecutarlo solo una vez.

7.1.3 Pruebas de Caja Negra

También conocidas como pruebas funcionales o de entrada y salida. Las pruebas de caja negra se centran en lo que se espera de un módulo, es decir, intentan encontrar casos en que el módulo no se atiene a su especificación. Por ello se denominan pruebas funcionales, y el probador se limita a suministrarle datos como entrada y estudiar la salida, sin preocuparse de lo que pueda estar haciendo el módulo por dentro. Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario (en sentido general: teclado, pantalla, ficheros, etc.), Es fácil obtener pruebas del 100% en módulos internos, aunque puede ser más laborioso en módulos con interfaz al exterior. En cualquier caso, es muy recomendable conseguir una alta cobertura en este tipo de pruebas. En conclusión una prueba de caja negra examina los aspectos del modelo fundamental del sistema sin tener mucho en cuenta la estructura lógica interna del software.

Si consideramos las definiciones anteriores para este sistema las pruebas a realizar son de caja negra ya que permiten saber cómo se comportara el sistema a través de la interfaz de usuario. Las pruebas que se harán se definen a continuación.

7.2 Tipos de Prueba: Pruebas de Sistema

Se harán pruebas al sistema completo con todas sus partes integradas ya que las pruebas al código o de caja blanca se fueron realizando a medida que se desarrollaba el software donde se corrigieron los errores en lógicos del sistema y en bucles que a medida que fueron apareciendo y como ya se ha dicho en la sección anterior se harán pruebas de caja negra donde se dará énfasis a la verificación de los datos de entrada y salida con el afán de comprobar que el sistema funcione correctamente y que las salidas de se correspondan con lo que se espera. Consecuentemente con lo que se ha planteado las pruebas de integración se han realizado cada vez que un nuevo modulo se ha agregado al sistema.

Prueba 1.1 (Ingresar Datos Profesores)
Actividades:

1. Ingresar un nuevo profesor con el nombre (*Profesor Uno*).
   1.2 Ingresar las preferencias horarias del profesor ingresado (*todas las horas*).
2. Cambiar el nombre del profesor por (*profesor dos*)

3. Cambiar las preferencias horarias del profesor
   4.1 Quitar todas las preferencias horarias del profesor
   4.2 Quitar algunas de las preferencias al azar.
4. Eliminar el profesor (*profesor dos*)
5. Verificar que la modificación de los datos sean correctos.

Resultados Esperados

1. Que el nombre del profesor sea incluido en el sistema
   1.2 Que las preferencias horarias sean incluidas en el sistema
2. Que el nombre ingresado como *Profesor Uno* posteriormente a ser modificado sea visualizado como *Profesor Dos*
3. Que el sistema no deje eliminar el total de preferencias horarias del profesor.
   3.1 Que el sistema muestre las nuevas preferencias horarias del profesor
4. Que el profesor seleccionado sea eliminado del sistema junto con sus preferencias horarias.

Prueba 1.2 (Ingresar Datos Salas)
Actividades:

1. Ingresar una nueva sala (*IBC 2-8*) con su capacidad de alumnos (*70*).
2. Cambiar el nombre de la sala (*IBC 2-9*).
3. Cambiar la capacidad de la sala (*90*).
4. Eliminar la sala ingresada.
5. Verificar que las modificaciones sean correctas.

Resultados Esperados

1. Que la nueva sala y sus capacidades sean incluidos en el sistema.
2. Que el nombre de la sala ingresada con anterioridad (*IBC 2-8*) sea modificado el sistema por (*IBC 2-9*).
3. Que la capacidad de la sala ingresada con anterioridad (*70*) se modificada por y se muestre la nueva capacidad (*90*).
4. Que la sala seleccionada (*IBC 2-9*) sea eliminada del sistema.

Prueba 1.3 (Ingresar Datos Cursos)
Actividades:

1. Ingresar un nuevo curso; sigla *(inf 1*), carrea (*ing.ejec.informática*), semestre (*1*), numero de cátedras por semana (*2*), cupo del curso*(50*), numero de ayudantías (*2*), nombre del curso(*informática 1*), cantidad de paralelos (*1*), profesor (*seleccionar de los profesores disponibles*), prerrequisitos (*no*).
2. Cambiar la sigla del curso ingresada (*inf 1*) por una nueva sigla (*ici 1*).
3. Cambiar la carrea (*ing.ejec.informática*) por una nueva carrera (*ing.civil.informática*).
4. Cambiar el número de cátedras por semana del curso (*2*) por (*3*) cátedras semanales.
5. Cambiar el cupo del curso (*50*) por un nuevo cupo (*70*).

6. Cambiar el nombre del curso (informática 1) por el nuevo nombre (*civilinformatica*)
7. Cambiar el profesor que dista el curso (*profesor que fue seleccionado*) por el profesor (*seleccionar un profesor de los disponibles distinto al que se encontraba seleccionado*).
8. Hacer combinaciones de los cambios especificados en los puntos anteriores.
9. Eliminar el curso que fue ingresado y luego modificado (*ici 1*).
10. Verificar que las modificaciones de los datos sean correctas.

Resultados Esperados:

1. Que el nuevo curso con todos sus datos (*sigla, semestre, número de cátedras, etc.*) sean incluidos al sistema y visualizados en la tabla cursos.
2. Que la sigla ingresada con anterioridad (*inf 1*) sea modificada en el sistema a (*ici 1*).
3. Que la carrera que había sido ingresada (*ing.ejec.informática*) cambie por la nueva carrera (*ing.civil.informática*).
4. Que el número de cátedras por semana (2) sea modificado en el sistema por el nuevo número de cátedras (3).
5. Que el cupo del curso (50) sea modificado en el sistema por el nuevo cupo para el curso (70).
6. Que el nombre original del curso (informática) cambien en el sistema por el nuevo nombre (civilinformatica).
7. Que el profesor designado para el curso (Eduardo Cartagena) sea modificado por el nuevo profesor para dictar las cátedras (Claudio cubillos).
8. Que las combinaciones de cambios que se quieran efectuar el sistema las realice sin ningún problema y los datos se reflejen en la tabla que muestra los cursos ingresados.
9. Que el curso seleccionado sea eliminado del sistema con todos sus datos.

Prueba 1.4 (Obtener Una Programación Horaria Válida)
Actividades:

1. Tratar de de obtener una solución sin haber ingresados datos al sistema.
2. Tratar de obtener una solución con datos en el sistema.
3. Modificar la solución obtenida.
4. Revisar la solución de acuerdo al horario y las salas están ocupadas.
5. Revisar los horarios que los profesores dictan las clases.
6. Correr sucesivamente el algoritmo y revisar cada uno de los pasos definidos con anterioridad.

Resultados Esperados:

1. El sistema envía un mensaje por la falta de datos.
2. Se genera un horario semanal válido con los datos ingresados para el semestre.
3. En caso de ser posible el cambio asignar el curso en el horario que el usuario designo o de lo contrario avisar a través de un mensaje que el cambio es inválido.
4. Se muestra un listado con todas las horas semanales con las salas que están ocupadas y en que horarios.

5. Se muestra los horarios en que cada profesor realiza alguna clase durante la semana.
6. Cada vez que se corra el algoritmo se deben mostrar los resultados y los pasos anteriores sin ninguna dificultad.

Prueba 1.5 (Verificar la Asignación de todos los cursos)
Actividades:

1. Generar una solución para el problema.
2. Revisar que todos los cursos ingresados se encuentren asignados.
3. Revisar que el número de ayudantías asignadas a cada curso se encuentren asignadas.

Resultados Esperados:

1. Una solución para el problema de asignación de cursos.
2. Que todos los cursos que se han ingresado con anterioridad se encuentran asignados en la solución obtenida.
3. Que las ayudantías correspondientes a cada curso se encuentren asignadas en la solución obtenida.

Prueba 1.6 (Verificar la Asignación en las Distintas Salas)
Actividades:

1. Generar una solución al problema.
2. Verificar que los cursos asignados a las distintas salas no excedan la capacidad de estas.
3. Verificar que todos los cursos asignados a la misma sala presenten el mismo color identificador.

Resultados Esperados:

1. Una solución para el problema.
2. Que los cursos asignados a cada sala tengan una menor cantidad de alumnos que la capacidad de la sala.
3. Que todas los curos asignados a una sala en particular presenten el mismo color identificador.

7.3 Resultados de Pruebas

Prueba 1.1 Ingresar datos Profesores

Se tomaron los datos y las acciones previstas por el plan de pruebas diseñado se ingreso el profesor tal como estaba descrito (*profesor 1*) se ingresaron todas la preferencias horarias para el profesor. Luego se cambió el nombre del profesor ingresado por (*profesor 2*) se cambiaron las preferencias horarias quitándolas todas y algunas al azar luego se eliminaron los datos ingresados.

Después de realizar todas las acciones descritas se confirmo que la prueba finalizó con éxito y que los datos se mantuvieron íntegros.

Prueba 1.2 Ingresar Datos Salas

Se tomaron los datos descritos en el plan de pruebas se ingreso la sala prevista por el plan con su capacidad respectiva luego se cambio el nombre de la sala y su capacidad, para posteriormente eliminar los datos ingresados.

Luego de todas estas acciones se pudo verificar que los datos se ingresaron, modificaron y eliminaron de forma satisfactoria.

Prueba 1.3 Ingresar Datos Cursos

Se tomaron los datos descritos por el plan de pruebas se ingreso un nuevo curso con todos los datos necesarios y requeridos por el sistema (*sigla, carrera, semestre, número de cátedras por semana, cupo, número de ayudantías, nombre del curso, cantidad de paralelos, profesor, prerrequisitos*) el curso se ingresó de manera satisfactoria visualizando los datos más descriptivos en la tabla destina para esto, se procedió a cambiar la sigla del cursos ingresado, la carrera la que pertenece el curso, el profesor y todos los datos que se describían en el plan de pruebas por separado y en cada una de los cambios descritos se realizo la modificación de manera exitosa. Luego se realizaron cambios combinando las distintas opciones de cambio y éstos se siguieron realizando de manera satisfactoria, eliminando el curso y confirmando que la eliminación, al igual que los cambios fuera exitosa.

Luego de realizar todas las acciones descritas en el plan de pruebas y verificar que los datos se mantuvieran íntegros, se estima que este ítem del plan termino de manera satisfactoria.

Prueba 1.4 Obtener Una Programación Horaria Válida

Se tomaron los datos y las acciones descritas en el plan de pruebas tratando de obtener una solución sin haber ingresados datos al sistema, lo cual no se pudo ejecutar puesto que el sistema aviso de esto a través de un mensaje. Se ingresaron datos al sistema y se ejecutó el programa para obtener la solución, del resultado se modificó la asignación de algunos cursos, se reviso que la solución no asignara cursos a los profesores en horarios que no tienen disponibles. Se realizaron las acciones descritas con anterioridad en diversas ocasiones.

Después de realizar este ítem del plan de pruebas descrito, se puede afirmar que la prueba se realizo de manera satisfactoria arrojando los resultados que se esperaban.

Prueba 1.5 Verificar La Asignación de Todos los Cursos.

Se tomaron las acciones descritas por el plan de pruebas generándose una solución al problema, a continuación se revisó que todos los cursos ingresados estuvieran asignados en la solución obtenida e inmediatamente después de haber terminado con la revisión de los cursos se verifico que los cursos que tenían ayudantías asignadas se encontraran en la solución.

Luego de haber realizado las acciones antes descritas se puede afirmar que este ítem del plan de pruebas se realizo de manara exitosa.

Prueba 1.6 Verificar la Asignación en las Distintas Salas

Se tomaron las acciones descritas en el plan de pruebas se genero una solución al problema después de los cual se procedió a revisar que cada uno de los cursos

asignados a las distintas salas disponibles tuviesen un cupo de alumnos menos o igual a la capacidad de la sala a la que fue asignado después de esto se procedió a verificar que todos os cursos asignados a la misma sala presentaran al mismo color identificador.
Luego de haber realizado las acciones descritas en este ítem del plan de pruebas y verificar que los resultados obtenidos se ajustan con lo esperado se puede afirmar que este ítem del plan finalizo con éxito.

Una vez realizados todos los ensayos explicados en el plan de pruebas y ejecutadas todas las acciones descritas en cada uno de los ítems se puede afirmar que se ha aprobado el plan de pruebas de manera exitosa, con el convencimiento que el sistema está en condiciones de ser entregado.

7.4 Pruebas de Eficacia del Algoritmo

En este capítulo se analizaran qué tan buenas son las soluciones que entrega el algoritmo de Búsqueda Tabú que se está utilizando para encontrar buenos costos; se ha analizado la calidad e los resultados comparándolos con los de otros algoritmos heurísticos.

Para esto se utilizará como referencia los resultados de la competencia internacional de timetabling PATAT[15] efectuada el año 2007.

Los organizadores de la competencia distribuyen los conjuntos de datos de entrada mediante un formato de texto estándar que los distintos algoritmos deben ser capaces de procesar, además de las especificación de un formato de entrega de resultados estándar y una utilidad que mide cuánto tiempo puede el algoritmo funcionar en determinada máquina (sólo máquinas PC, uniprocesador). Para esto se adaptó un ejecutable c que hace uso de la librería tabú sin cambio alguno. Luego de entregado el resultado se utiliza un programa, también distribuido por la competencia que evalúa la factibilidad y las violaciones de restricciones que presenta. Los costos arrojados por esta utilidad son los utilizados en estas pruebas.

Los resultados que se muestran a continuación son una comparación de las violaciones de restricciones duras (eje vertical) en 3 conjuntos de datos, los algoritmos fueron ejecutados 10 veces para cada conjunto de datos (eje horizontal).

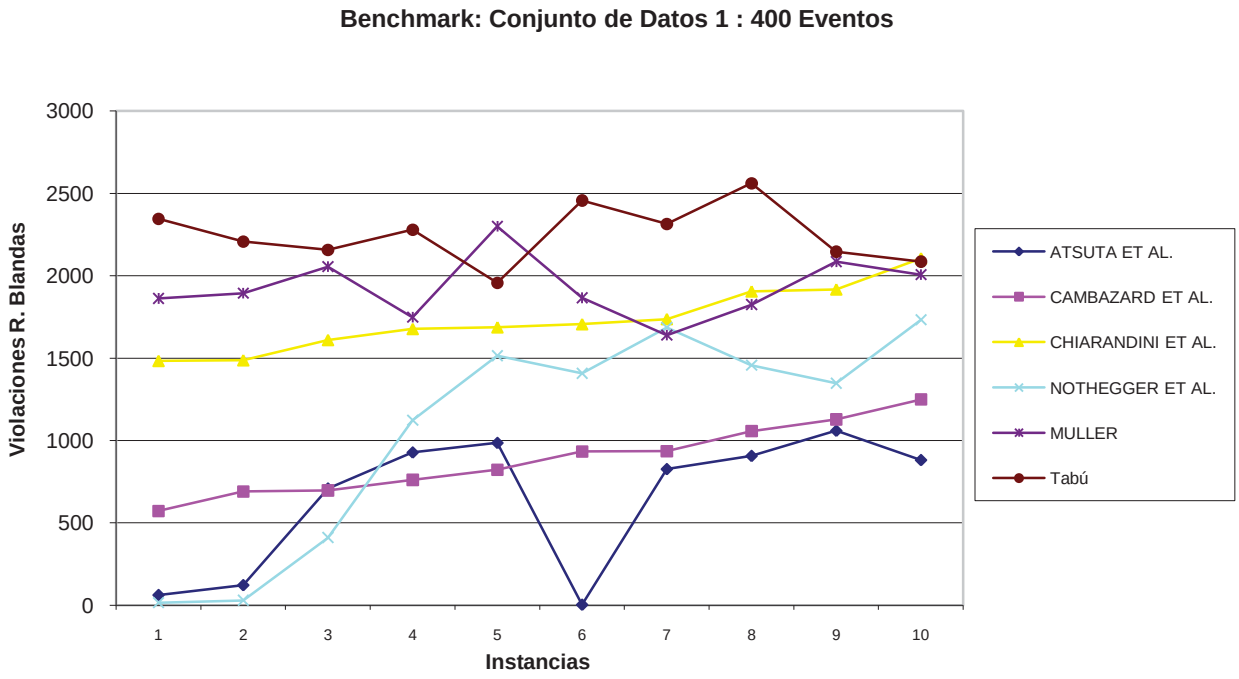
La curva rotulada como Búsqueda Tabú corresponde a los resultados obtenidos para esta implementación, las otras 5 curvas corresponden a los 5 primeros lugares en la competencia para la categoría de post enrollment timetabling, también conocido como timetabling universitario que es el problema tratado en esta investigación.

Es de vital importancia notar que debido a las diferencias existentes en el conjunto de restricciones blandas evaluado en la competencia y las restricciones blandas para las cuales el presente algoritmo fue diseñado, no existió una correspondencia total en el criterio de intensificación, por lo que los costos obtenidos fueron de una clara menor calidad.

7.4.1 Gráficos

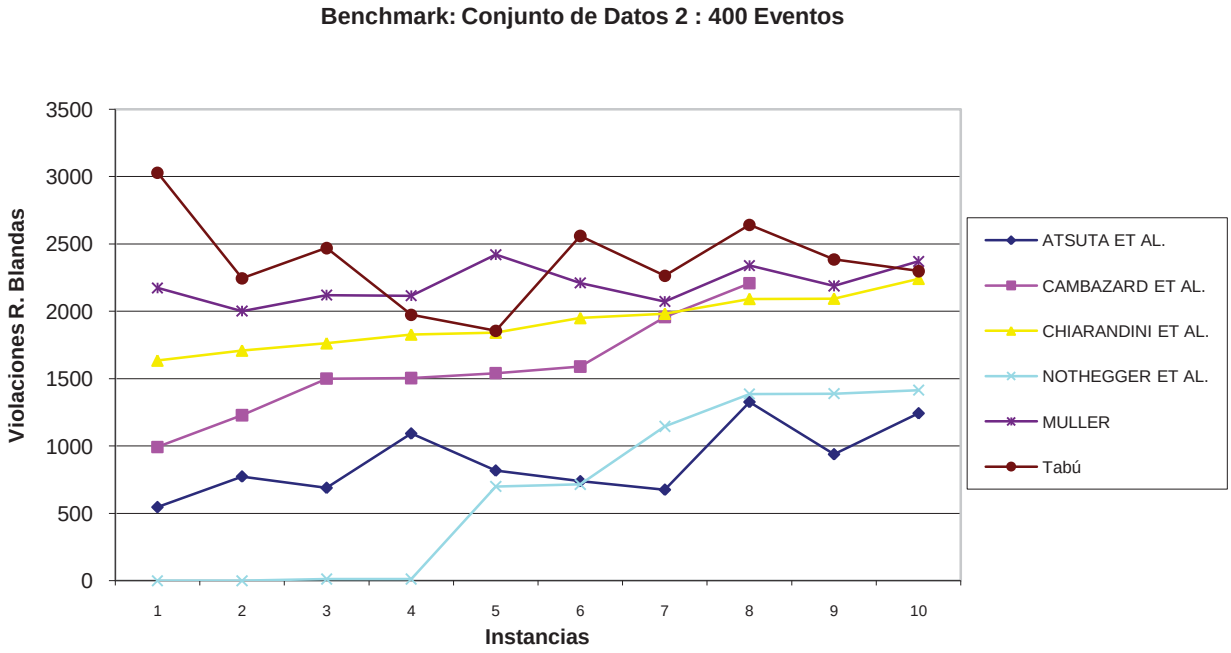
Los tipos de algoritmos utilizados por los competidores se detallan a continuación :

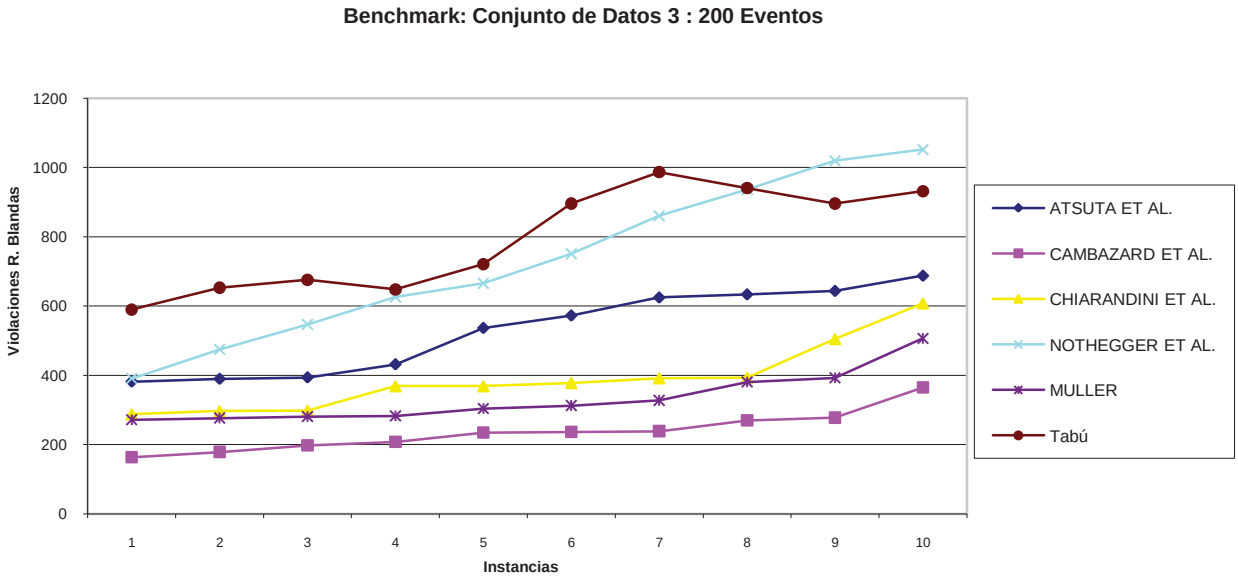
- **ATSUTA ET AL.:** Búsqueda Tabú / Búsqueda iterativa local
- **CAMBAZARD ET AL.:** Simulated Annealing / Método Húngaro
- **CHIARANDINI ET AL.:** Tabú Search
- **NOTHEGGER ET AL.:** Ant Colony Optimización (ACO)
- **MULLER :** Hill Climbing / Great Deluge
- **TABÚ :** El algoritmo presentado en esta investigación



En estos resultados se ve reflejado el desempeño de todos los métodos mayores que existen para la resolución del problema del timetabling universitario. Los mejores resultados pertenecen al algoritmo propuesto por ATSUTA et al, quien utilizó un solver de propósito general para problemas de satisfacción de restricciones (constraint satisfaction problem), que utiliza una combinación de búsqueda local iterativa y búsqueda tabú para resolver un conjunto de desigualdades y restricciones, cuidadosamente ajustadas para tender a encontrar soluciones.

La mayoría de los métodos de solución propuestos en esta competencia utilizan alguna mezcla de métodos estocásticos para ayudar en la búsqueda de óptimos locales, junto con metaheurísticas para moverse por distintos vecindarios de soluciones.





El algoritmo presentado en este trabajo, consta con una fase inicial de construcción en la que se realiza una asignación aleatoria de eventos (cursos) en posiciones factibles, muchas veces quedan eventos sin asignar, ya que solamente quedan posiciones disponibles que violan restricciones duras. Para solucionar esto se realizan intercambios de eventos

(swap o flip), eligiendo solamente los pares que mantienen la factibilidad de las asignaciones, el intercambio puede ser entre dos eventos asignados o entre un evento asignado y una posición vacía (reasignación del evento). En otros trabajos, particularmente bajo las condiciones de la competencia PATAT, se opta por permitir comenzar la optimización del costo (restricciones blandas) basándose en una solución inicial parcial, en donde no todos los eventos se encuentran asignados, ya que debido al diseño del algoritmo se pueden ir incorporando los distintos eventos sin asignar a medida que se itera para minimizar los costos. También en algunos casos se permite llegar a soluciones incompletas o conjuntos de asignaciones donde se violan restricciones duras, esto a favor de métodos muy veloces que pueden manejar un número de soluciones posibles mucho mayor que en casos donde se mantiene una estricta consistencia de la solución con la que se está trabajando, en cuanto a restricciones duras. En el caso del algoritmo aquí presentado se invierte una cantidad significativa de tiempo en encontrar una solución inicial bajo la cual iterar, para buscar distintos óptimos locales sin violar restricciones duras.

No obstante en los resultados de la competencia anteriormente expuestos existen métodos de resolución fundamentalmente muy similares en este aspecto, que obtienen consistentemente un margen superior en la calidad de las soluciones encontradas.

Al estudiar los algoritmos mencionados queda en evidencia que la simpleza del diseño aquí mostrado podría tener un papel perjudicial en los resultados obtenidos, principalmente en las fases de diversificación e intensificación, en las que los intercambios pseudo-aleatorios e intercambios de minimización de costo aplicados de una manera voraz, respectivamente podrían mejorar significativamente con la introducción de un método estocástico de búsqueda de óptimo local más elaborado, como podría ser el método húngaro y de la aplicación de una segunda técnica metaheurística, como simulated annealing, para apoyar la fase de diversificación de soluciones.

Junto con lo anterior mencionado, que se propone como base para un posible trabajo futuro, es importante mencionar el ajuste de parámetros, que se cree ha sido crucial para entender los resultados obtenidos en las diferentes pruebas, y que es susceptible de ser mejorado usando los resultados aquí obtenidos. Otra forma en la que se podrán mejorar los resultados obtenidos es fusionar el algoritmo de Búsqueda Tabú desarrollado con alguna otra metaheurística de las que se han mencionado en este trabajo.

CAPÍTULO 8:
Conclusión

El problema de asignación de horarios es un problema difícil de enfrentar por la gran cantidad de variables que intervienen en él. El objetivo de esta investigación fue estudiar y entender el problema, centrándose en el caso particular de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso.

Además de entender al problema se debió investigar, estudiar y comprender los distintos métodos existentes para la resolución del mismo. Entre las variadas metaheurístcas que se investigaron se optó por elegir la Búsqueda Tabú, la que en esencia busca acercarnos, en un problema donde intervienen gran cantidad de variables, sino a la mejor solución, a una que esté muy cerca de serlo. Esto se logra a través de los mecanismos que se definen, como pueden ser la definición de una vecindad, o el uso de memoria para definir los movimientos tabú, entre otros que fueron expuestos en detalle en este informe.

Junto con lo anterior, se produjo toda la documentación necesaria para poder desarrollar el software. Estos documentos (casos de uso, diagramas de secuencia, diagramas de flujo) fueron la guía por medio de la cual se implementaron los distintos componentes necesarios para poder desarrollar un software de calidad, capaz de entregar una solución acorde con las necesidades de la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso, y en concordancia con la investigación realizada.
En concordancia a lo anterior se pudo terminar con la etapa de desarrollo del software tanto del algoritmo que entrega las soluciones al problema de asignación de cursos como de la interfaz gráfica, necesaria para una interacción fluida con el usuario, ingresando datos de entrada, y mostrando las soluciones obtenidas a partir de la implementación de la metaheurística. Gracias a esto se está en condiciones de entregar un software completamente funcional.

Junto con esto, y adicionalmente se propone para un posible trabajo futuro una revisión de metaheurísticas y métodos estocásticos de búsqueda de óptimos locales, adicionales para mejorar las fases de diversificación e intensificación respectivamente en el algoritmo presentado. También se propone la revisión y optimización del ajuste de parámetros actual, posiblemente utilizando un conjunto de datos de prueba y una herramienta de ajuste de parámetros estandarizada como lo es paramILS[16].

Se espera que el producto entregado a la Escuela de Ingeniería Informática de la Pontificia Universidad Católica de Valparaíso sea una herramienta que facilite en gran medida la tarea de realizar la asignación de cursos semestralmente.

CAPÍTULO 9: Referencias

[1] Pilar Moreno Díaz, Gabriel Huecas Fernández-Toribio, Jesús Sánchez Allende, Almudena García Manso. Metaheuristicas de Optimización Combinatorial: Uso de Simulated Annealing para el Problema de Calendarización.

[2] Juan Enrique Molina Araya .algoritmos evolutivos para el problema de tipo timetabling.

[3] Oscar A. Chávez Bosquez, Guillermo de los Santos Torres, José Luis Gómez Ramos. Búsqueda Tabú aplicada al problema NP completo.

[4] Belén Melián, José A Moreno Pérez , J. Marcos Moreno Vega. Metaheuristics: A global view.

[5] Franklin Johnson Parejas. Marco de Trabajo de hipercubos para ACO.

[6] Carlos A. Coello Coello : Búsqueda Tabú . Evitando lo prohibido.

[7] R. Álvarez-Valdési , E Crespo, J.M Tamariti :Experiencias de Utilizar Búsqueda Tabú la resolución de problemas.

[8] Whitten, Jeffrey L.; Lonnie D. Bentley, Kevin C. Dittman. (2004). *Systems Analysis and Design Methods*. 6th edition. ISBN 025619906X.

[9] Conrad Weisert, Waterfall methodology: there's no such thing! : http://www.idinews.com/waterfall.html [Visitado 15/11/2009]

[10] Ivar Jacobson, Grady Booch, and James Rumbaugh (1999). *The Unified Software Development Process*

[11] Extreme Programming: http://c2.com/cgi/wiki?ExtremeProgramming [Visitado 15/11/2009]

[12] The Agile Unified Process: http://www.ambysoft.com/unifiedprocess/agileUP.html. [Visitado 15/11/2009]

[13] Roger Pressman: Software Engineering: A Practitioner’s Approach

[14] Fernando Lopez Hernandez: JNI .java native interface.

[15] http://www.cs.qub.ac.uk/itc2007/index.htm. [Visitado 15/11/2009]

[16] paramILS: herramienta de ajuste de parámetros para algoritmos http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/ [Visitado 30/06/2010]