



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

REDISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECUENTO DE UNIDADES
DOCENTES PARA LA FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

WILLY ADOLFO MAIKOWSKI CORREA

PROFESOR GUÍA:
PATRICIO POBLETE OLIVARES

MIEMBROS DE LA COMISIÓN:
JAVIER VILLANUEVA GONZÁLEZ
LUIS MATEU BRULE

SANTIAGO DE CHILE
DICIEMBRE 2015

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: WILLY ADOLFO MAIKOWSKI CORREA
FECHA: DICIEMBRE 2015
PROFESOR GUÍA: SR. PATRICIO POBLETE OLIVARES

**REDISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE RECUENTO DE UNIDADES
DOCENTES PARA LA FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS**

El recuento de Unidades Docentes (UD's) es una de las tantas herramientas dentro de la plataforma U-Campus, un sistema desarrollado por el Área de Infotecnologías (ADI) en respuesta a la necesidad de facilitar el cumplimiento de los servicios entre alumnos y funcionarios de la Universidad de Chile. El recuento de UD's forma parte de los pasos necesarios para corroborar el avance académico del alumno y está inserto dentro de los procesos necesarios para realizar la titulación o graduación.

Antiguamente no se poseía esta herramienta y, por ende, se debía realizar un chequeo manual con planillas creadas para cada tipo de título. Esta verificación tomaba meses en ser calculada y, debido a ello, hubo varios intentos por crear un sistema que lo apoyara. Sin embargo, sólo logró una mejora significativa cuando se desarrolló el sistema por el ADI. Esta implementación consiste en recorrer todas las combinaciones posibles hasta que se encuentre la solución o hasta que se cumpla un plazo de quince segundos dejando inconclusa la certeza de si la solución obtenida hasta ese momento corresponde a la mejor. Debido a lo anterior, se producen casos en que el recuento de UD's responde incorrectamente, ya sea mencionando que una persona no cumple con los requisitos académicos, siendo que si los satisface (falsos negativos), o entregando una nota inferior a la que una combinación encontrada posteriormente podría entregar.

Inspirado en los problemas que provoca esta falta de exactitud para el agente principal de este proceso, la Subdirección de Gestión Docente (SGD), y en conceptos de programación dinámica y teoría de grafos, se diseñó una mejora en la heurística realizando reducciones en el número de combinaciones. El modelamiento como grafo permitió encontrar componentes conexas que reflejaron la independencia que tienen ciertas combinaciones con respecto a otras, permitiendo una separación inicial en problemas más pequeños que posteriormente son adicionados para componer la solución general.

La mejora impactó de una manera considerable en los tiempos, reduciéndolos hasta diez veces, al igual que el número de cálculos que se demoraban más de quince segundos. Incluso se encontraron alumnos ya egresados con la posibilidad de tener una mejor nota debido al nuevo cálculo. Aun así, no se logró uno de los requisitos de calidad del objetivo principal para la carrera Ingeniería Civil Industrial, debido a la oferta de ramos electivos que posee. La gran mayoría son de cinco créditos (la mitad de lo usual en la Escuela), produciendo que un alumno promedio deba aprobar el doble de ramos que si fuese de otra carrera, afectando negativamente al algoritmo. Por lo anterior, se complementó la conclusión con la necesidad de herramientas que sincronicen los esfuerzos en la formación de innovaciones dentro de los planes de estudios por parte de los departamentos, tanto con el área de gestión por parte de la Escuela como del área tecnológica competente, como lo es el ADI.

*“... Te escribí todas las noches, lector,
para dejarte entrar en mi mundo, porque
no lo conoces, pero eres parte de él.*

Y también te escribí a ti.”

Francisca Solar, #100writingdays

Agradecimientos

Gracias a mi familia, a mis padres y hermana por guiarme hacia este camino desde pequeño e inculcarme la importancia del saber. También a Daniela y nuestra hija Emily, por la comprensión y tolerancia, y por ser la alegría al llegar a casa, siendo el motor de motivación en estos últimos años.

Gracias al ADI, a Manuel por ser un apoyo importante para esta memoria preocupándose de manera desinteresada. También a Javier, por la oportunidad y la visión, si no fuera por él quizás donde hubiese encontrado mi lugar en la universidad (o fuera de ella). A Víctor y al equipo entero en realidad, por confiar y hacerme parte, por aceptarme y aguantarme.

Gracias a Patricio Poblete, profesor guía de esta memoria, por la confianza, dedicación y el tiempo regalado. A mis amigos Fernando, Mauricio y Paulina, por la compañía en la U por este largo proceso y por la preocupación en mi desarrollo.

Finalmente, quiero agradecer a todas esas personas que han compartido un pequeño momento de sus vidas conmigo, ya que quizás sin saberlo también aportaron a mi crecimiento y contribuyeron en esta memoria. Además, gracias a ti lector, por tan sólo leerme.

Tabla de Contenidos

Introducción	1
1. Descripción del Proyecto	3
1.1. Antecedentes	3
1.2. Motivación	6
1.3. Objetivos	9
1.4. Metodología	9
2. Marco Teórico	11
2.1. Complejidad	11
2.2. Teoría de Grafos	12
2.3. Optimización Combinatorial	14
3. El Problema: Situación Actual	17
3.1. Combinaciones	17
3.2. Proceso Presente	20
4. Solución: Modelo y Rediseño	22
5. Implementación	28
6. Validación	31
Conclusión	35
A. Código de la Implementación	39

Índice de Tablas

6.1. Tabla con los códigos de carrera y sus respectivos nombres.	31
6.2. Tabla de resultados de la heurística antigua.	32
6.3. Tabla de resultados del algoritmo nuevo.	33

Índice de Ilustraciones

1.1.	Procesos necesarios para la obtención de un Título Profesional	4
1.2.	Árbol de planes y ramos con sus respectivas reglas y unidades	5
1.3.	Planes de la carrera de Ingeniería Civil en Computación (v3).	7
1.4.	Planilla base ocupada para el desarrollo de la memoria descrita en [2]	8
2.1.	Gráfico con las curvas de órdenes de complejidad en una escala logarítmica .	12
2.2.	Ejemplo de grafo dirigido.	13
2.3.	Ejemplo de componentes conexas.	13
2.4.	Árbol de decisión para ordenar tres elementos A, B y C.	14
2.5.	Cálculos hechos en serie de Fibonacci siguiendo perspectiva top-down. En rojo se puede observar los cálculos necesarios si se siguiera una perspectiva bottom-up.	16
3.1.	Imagen del módulo de titulación donde se muestran los procesos necesarios para comprobar los requisitos académicos.	20
3.2.	Estadísticas de peticiones por día relacionadas al recuento de UD's.	21
4.1.	Árbol de planes con sus respectivas reglas como etiqueta de los nodos.	23
4.2.	Conversión del problema en un grafo.	24
4.3.	Árbol de reglas para la carrera Ingeniería Civil Electricista (v2).	25
4.4.	Extracto de la figura 4.2 para evidenciar la recursividad de las componentes.	26
4.5.	Comportamiento del caché. En rojo el lugar donde se ubica el ramo y en blanco la componente donde no se posee. En gris los cálculos que pueden ser obtenidos desde la memoria caché ya que ya fueron calculados anteriormente.	26
5.1.	Funciones de la implementación anterior y sus comportamientos.	29
5.2.	Funciones de la nueva implementación y sus comportamientos.	29
6.1.	Grafo de un alumno de la carrera Ingeniería Civil Industrial.	34

Introducción

Si uno se preguntara cómo corroborar si una persona cumple todos los requisitos académicos para aprobar una carrera se pensaría en los elementos principales que son los ramos. También, las universidades disponen de una malla desde que el estudiante postula y uno podría pensar que chequear estos requisitos consistiría netamente en contrastar los ramos con esta malla. Bueno, eso sería si no existiera ninguna flexibilidad definiendo todos los ramos como obligatorios. Aun así, es común encontrar que estas mallas se dividen en líneas específicas con diferentes objetivos, por ejemplo, las carreras de ingeniería poseen una línea de ramos matemáticos y otra línea de ramos humanistas para complementar el desarrollo del alumno con habilidades de todo ámbito. Eso sí, no se definen todos estos como obligatorios, dejando una lista de ramos como opción para aprobar cada una de estas líneas. Es decir, una malla esta compuesta tanto por elementos que deben ser aprobados obligatoriamente, como por otros que son opcionales.

Pensando en la Universidad de Chile, y más específicamente en la Facultad de Ciencias Físicas y Matemáticas (FCFM), se posee aun mayor flexibilidad debido a que se pueden contabilizar incluso ramos de distintas carreras. Esto debido a que se ingresa a un plan común y después de dos años se puede ir escogiendo los ramos para su especialidad. Por esto, para cuando un alumno ya se encuentra en la etapa de titulación, para todos los ramos se deben ver todas las opciones donde puede ser ubicado y cuál combinación permite verificar que el alumno cumple todos los requisitos académicos.

Ahora bien, los ramos pueden poseer distinta exigencia y estas mallas pueden requerir la aprobación de ramos dependiendo de ello. Es común entonces encontrar que los ramos son medidos en créditos utilizándolos como medida de comprobación de la dificultad y como límite para que un alumno no sobrepase cierta exigencia por semestre. Para la Universidad de Chile estos créditos son conocidos como Unidades Docentes (UD's).

Antiguamente, en la FCFM para la verificación de los requisitos académicos se realizaba este procedimiento manualmente. El Secretario de Estudios con el conocimiento de realizar este proceso cada semestre generó planillas que lo ayudaban, pero aún así la verificación tomaba meses. A partir del año 2013, y luego de varios intentos por parte de las autoridades, el Área de Infotecnologías (ADI) implementó un sistema de recuentos automático para apoyar esta labor manual, el que implicó una mejora considerable, permitiendo que la misma labor pasara de realizarse de meses a segundos.

En esta solución se modeló la malla en un árbol de planes, siendo estos parecidos a las líneas mencionadas en el primer párrafo. Sin embargo, aunque se logró un gran avance, sigue

siendo posible realizar mejoras debido a que el chequeo no está siendo abarcado en su totalidad existiendo falsos negativos y entregando notas inferiores al máximo. Por el problema que esto genera, por el desafío que constituye y por el impacto que puede provocar una nueva solución y entendimiento de este caso, se motivó esta memoria, buscando como objetivo principal lograr la implementación de un sistema que elimine en lo posible los falsos negativos, aumentando la confiabilidad y encontrar casos de mejora en la nota de titulación del alumno.

A continuación, se explicitarán todos los pasos necesarios, tanto para comprender con mayor detalle el problema, como la solución que se encontró. En el capítulo 1, se entregará un contexto previo y una explicación más profunda de los objetivos buscados y la metodología efectuada. En el segundo capítulo se introducirán conceptos teóricos que no sean de conocimiento general y que sirvan para el entendimiento de la memoria. En el capítulo 3, se explicará el problema y a los actores involucrados, permitiendo explicar la solución propuesta en el capítulo 4. Posteriormente, en los capítulos 5 y 6 se explicitará de manera breve cómo se llevó a cabo la implementación y la validación efectuada para corroborar que la solución presentase mejoras significativas. Finalmente, en el capítulo 7 se concluirá con la interpretación de los resultados y con propuestas futuras.

Capítulo 1

Descripción del Proyecto

A continuación se presentarán los conceptos básicos necesarios para la comprensión de la memoria. Primero, se introducirá la plataforma en la que está alojada la solución actual para, luego, explicar paso a paso el módulo y sus elementos.

Después, se profundizará en el elemento principal: los planes. Para ello se mostrará su estructura, reglas y alcances. Además, se exemplificará una carrera para mostrar cómo se realiza la comprobación de satisfacción y el cálculo del promedio.

Posteriormente, se profundizará en los elementos que motivaron esta memoria y la relación que poseen en conjunto con los objetivos.

Finalmente, se mencionará concisamente la metodología seguida para afrontar el problema y la implementación de la solución.

1.1. Antecedentes

El Área de Infotecnologías (ADI) es un equipo de personas dedicadas a dar soluciones tecnológicas a la Universidad de Chile. Entre las soluciones que ha presentado se destacan dos plataformas: U-Cursos y U-Campus.

La primera, está orientada al apoyo docente, siendo un nexo entre el alumno y el profesor. Es decir, permite que existan foros para relacionarse, secciones de material docente para compartir documentos, un lugar para publicar las notas que va obteniendo el alumno en sus evaluaciones, entre otros.

En cambio, U-Campus está orientado a lo administrativo. Hay variados procesos que se dan en la universidad en los que existe interacción entre un alumno y los funcionarios no académicos. Un ejemplo de ello era la entrega de certificados, en donde un alumno se dirigía a un funcionario para su solicitud.

Aun así, una amplia cantidad de estos procesos han pasado por un periodo de automatización de manera tal de facilitar la parte operativa y orientar la administración a velar por el cumplimiento estratégico de la universidad. Entre estos procesos se encuentra la titulación. Ésta consiste en variadas etapas que distintas áreas de la facultad deben verificar de manera tal de chequear finalmente si un alumno está habilitado para obtener un título profesional.

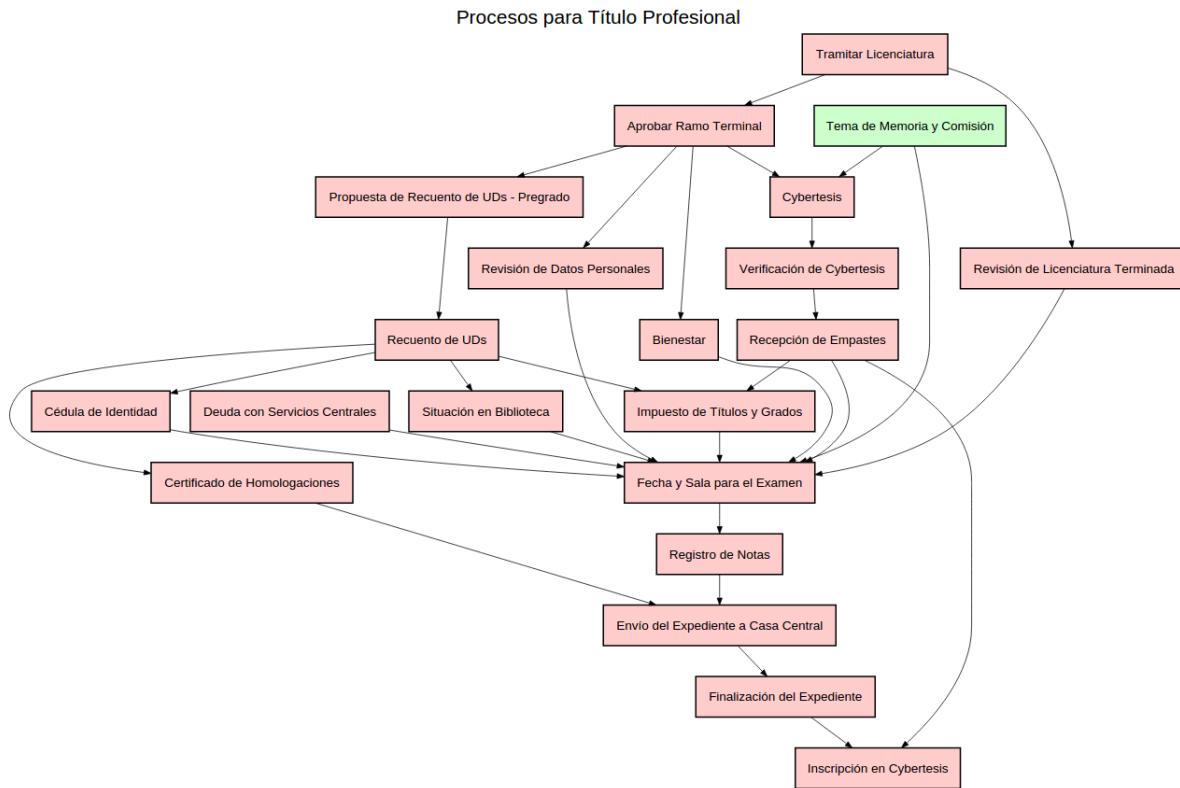


Figura 1.1: Procesos necesarios para la obtención de un Título Profesional

Una de las etapas de verificación consiste en chequear si un alumno posee el avance académico suficiente. Este procedimiento es conocido como el “Recuento de UD’s” debido a que un alumno debe ir aprobando cursos que son medidos en “Unidades Docentes” (unidad de medida de demanda horaria) de tal manera que se cumpla una serie de reglas. Estas reglas antiguamente consistían netamente en sumar una cantidad específica de UD’s y de allí nació la palabra recuento.

En la antigüedad, este procedimiento se hacía manualmente y tomaba meses en ser calculado. Anteriores memorias intentaron lograr que este procedimiento fuera cada vez mejor, pero no fueron fructíferas. Por ejemplo, en “Diseño de un sistema inteligente de recuento de Unidades Docentes para la Escuela de Ingeniería y Ciencias” [5] sólo se logró el entendimiento en profundidad del problema proponiendo una solución y en “Diseño y construcción de un sistema de apoyo a la verificación del cumplimiento del plan de estudios en la Facultad de Ciencias Físicas y Matemáticas” [2] se logró llegar a un sitio web que sólo reemplazaba lo que antiguamente se hacía en papel por un formulario web, sin modificar el proceso.

Finalmente, hace algunos años en U-Campus se agregó un módulo dedicado para ello que centralizaba toda la información del alumno con sus cursos aprobados. El módulo acogió el

nombre de “Recuento de UD’s” y logró minimizar los tiempos de meses a segundos.

Para lograr evaluar el avance académico de un alumno se debió modelar un título profesional en planes. Un plan es un conjunto de ramos y/o subplanes con una regla específica. Dado que poseen subplanes y estos, a su vez, también pueden poseer subplanes se empieza a armar un estilo de árbol en donde el plan principal sería la cúspide, siendo el último en ser evaluado, debido a que para cumplirlo se debería concluir previamente los de más bajo nivel, por ende, a este plan se le conoce como la carrera (si es que hablamos de título profesional).

Las reglas mencionadas previamente pueden ser tres: regla contar, UD’s o todo. La regla contar quiere decir que para concluir el plan se debe aprobar cierta cantidad de ramos. Si un plan con esta regla posee, a su vez, un subplan con la misma regla, los ramos aprobados del subplan son adicionados al contador del plan principal. La regla UD’s es similar, pero en vez de contar cantidad de ramos aprobados lo que se hace es sumar las unidades docentes de los ramos aprobados.

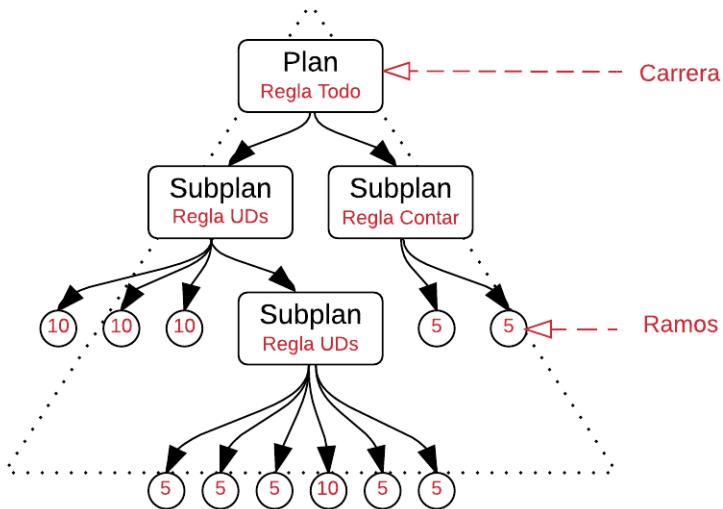


Figura 1.2: Árbol de planes y ramos con sus respectivas reglas y unidades

En la figura 1.2 se puede observar el árbol de planes y ramos con sus reglas y unidades, respectivamente. Para el caso del plan de más bajo nivel que posee regla UD’s, si ésta fuera sumar veinticinco, se podría tomar una combinación de ramos que dé esa cantidad, por ejemplo, todos los ramos de cinco unidades. Para el plan con regla contar, si ésta fuera de contar una unidad se debe aprobar al menos uno de sus ramos sin importar las unidades que poseen.

Dado que las reglas son de contar o sumar alguna unidad, existe la forma de no cumplir un subplan. Si el plan del segundo nivel que posee regla UD’s tuviera que sumar treinta unidades, éstas podrían sumarse a partir de los ramos que posee sin necesidad de sumar las unidades de su subplan. De esta manera el subplan de más bajo nivel se daría por cubierto, aunque no se cumpla su regla.

Aun así, existe la regla “todo” que no permite el comportamiento anterior, debido a que obliga a cumplir todos los elementos (ramos y subplanes) que pertenecen a un plan. En la

imagen anterior (figura 1.2), la carrera posee la regla todo y, por ende, los dos subplanes que contiene deben cumplirse de manera obligatoria.

En la Facultad de Ciencias Físicas y Matemáticas el módulo recuento de UD's es ampliamente ocupado y las carreras que han sido modeladas poseen variedad de combinaciones de las reglas antes descritas. Además, se poseen versiones debido a que en ciertas generaciones los cursos que se dictan son distintos, siendo un hecho natural debido al avance académico y a las innovaciones que se realizan a la malla curricular año tras año.

Un cambio de versión implica referencias complejas dentro de los ramos, debido a que dos ramos en una versión pueden unirse en uno a la fase siguiente, implicando diferencias en la cantidad de UD's, la nota que debe ser puesta, entre otros. Es decir, un alumno que poseía esos dos ramos antiguos aprobados y pasa a la nueva versión del plan tendrá inconsistencias en esos valores. Por lo anterior, el manejo de versiones en los casos complejos no se considera dentro del alcance de esta memoria, pero se abarcará las equivalencias uno a uno que puedan poseer los ramos.

Si observamos cómo ha sido modelada la carrera “Ingeniería Civil en Computación” tendremos un ejemplo claro del diseño estándar dentro de la facultad. A continuación se muestra un diagrama con la última versión, donde se evidencia la tenencia de una línea de cursos obligatorios y otra de cursos electivos.

Un punto importante de la titulación es la nota con la que un alumno egresa. Para realizar ese cálculo se debe tener en consideración el tipo de título que se obtiene debido a que es posible promediar todos los ramos o, si es título profesional, promediar sólo los ramos de los planes de carrera ([1]). En el caso de la figura 1.3, se tendría un promedio con las notas de los planes que están superior a la línea roja punteada, pero en el caso de otro tipo de título se deberían promediar todos los ramos.

Por todo lo antes descrito, una correcta solución debe velar por satisfacer dos condiciones: determinar si un alumno está en momento de titularse y encontrar la mejor nota para ello. La solución actual no confirma del todo estas dos condiciones, ya que se tienen falsos negativos (puede mencionar que un alumno no puede titularse siendo que sí posee todo el avance curricular necesario) y no se tiene garantía de que entregue la mejor nota.

1.2. Motivación

Si bien actualmente se tiene una mejora considerable de lo que es la titulación, en lo que respecta a la propuesta de recuento de unidades docentes, aún se sigue con un procedimiento de chequeo manual. Durante el tiempo ésta se ha ido acortando cada vez más, pero un desarrollo mayor en esa área significaría el punto culmine logrando obtener resultados confiables y en un tiempo considerablemente mejor.

Además, el desafío es de gran envergadura lo que incentiva solucionarlo. Como se mencionó en los Antecedentes, hubo dos memorias en donde se intentó llegar a una mejora automática de este proceso. En [5] el objetivo principal era lograr un sistema lo más automatizado

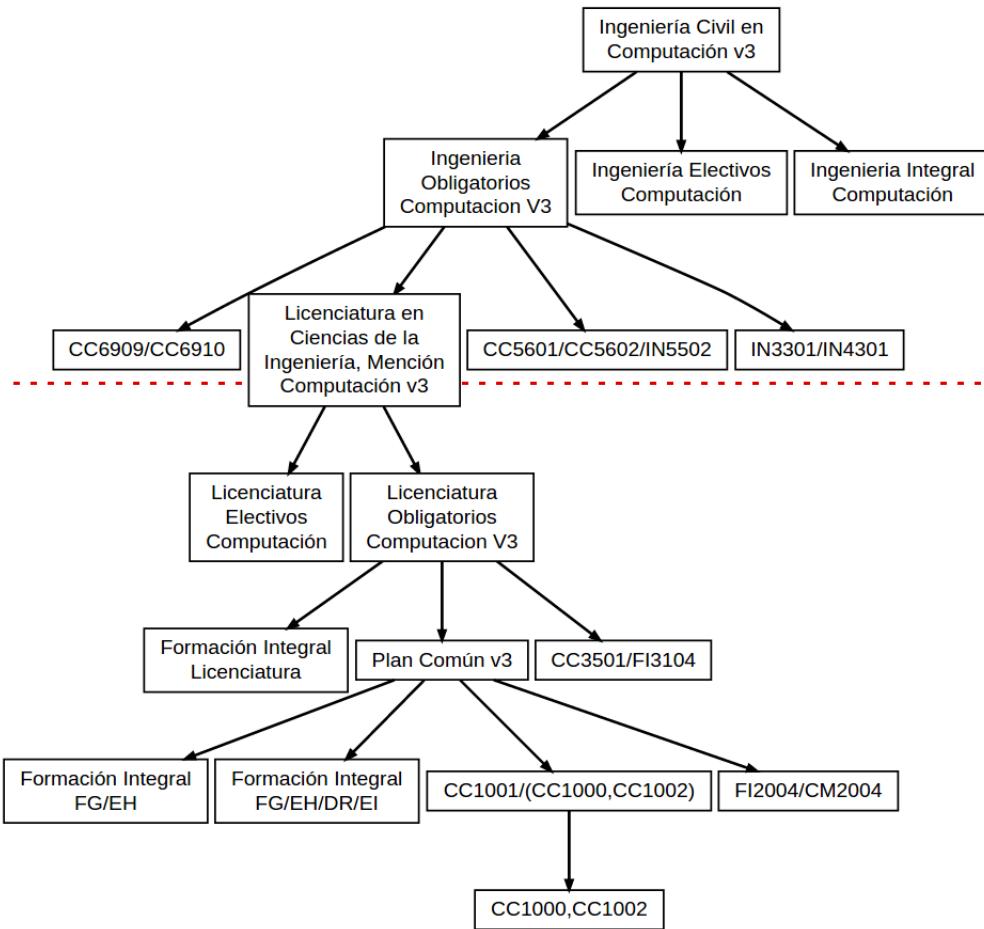


Figura 1.3: Planes de la carrera de Ingeniería Civil en Computación (v3).

possible pero sólo se logró una definición teórica de él. Aun así, se avanzó ampliamente en la definición observando el proceso realizado por el Secretario de Estudios que mediante planillas lograba evaluar el avance curricular del alumno. Además, en este trabajo se propusieron requisitos funcionales para una aplicación final ya que se llevó a cabo un estudio de los decretos universitarios desde 1985.

Por otra parte, en [2] se tenía presente la anterior memoria, por lo que se logró avanzar en la automatización, pero a partir de una perspectiva de apoyo netamente y no de reemplazar parte del proceso. Esto quiere decir que se construyó un sitio en el que la persona encargada debía seguir los mismos pasos que si lo hiciera con papel. Si bien se podían centralizar datos y se contaba con una base de datos donde se guardaban los registros de recuento, no logró mayor mejora.

Sistema de Recuento de Unidades Docentes - Resultado del Recuento - Mozilla Firefox	
Archivo	Editar
NOMBRE:.....	AÑO INGRESO:.....
Nº MATRICULA:.....	RUT:.....
SISTEMA DE RECUENTO DE UNIDADES DOCENTES	
Resultado del Recuento	
PLAN COMUN FACULTAD: 242 UD.	ELECTIVOS LIC.: 28 UD.
LICENCIATURA: 140 UD.	LICENCIATURA: 140 / 140 U.D.
10 CC 30A 20 FI 10A 09 FI 21A 09 FI 21B 09 FI 22A 05 FI 25A 09 FI 33A 09 FI 34A 05 FI 35A 20 MA 11A 20 MA 12A 09 MA 22A 09 MA 25A 09 MA 26B 09 MA 33A 09 MA 34A 05 SU 20A 09 QH 21A 00 IDIOMA 09 EH	10 CC 30A 10 CC 31A 10 CC 31B 10 CC 40A 10 CC 41A 10 CC 41B 10 CC 41C 10 CC 42A 10 IN 34/INMA 3/A 10 IN 41A 10 IN 42A 10 IN 46A 10 MA 34B 09 MA 33A 09 MA 34A 05 SU 20A 09 QH 21A 00 IDIOMA 09 EH
PLAN COMUN FACULTAD: 242 / 242 UD.	LICENCIATURA: 140 / 140 U.D.
10 CC10A 0 EH09A 9 EH22A 9 EH22E 20 FH10A 9 F121A 9 F121B 9 F122A 5 F125A 9 F133A 9 F134A 5 F135A 20 MA11A 10 MA12A 0 MA22A 0 MA25A 0 MA26B 0 MA33A 0 MA34A 0 SU20A 0 QH21A 0 IDIOMA 0 EH	5,5 ✓ 7 ✓ 5,4 ✓ 5,3 ✓ 4,6 ✓ 4 ✓ 5,8 ✓ 4,7 ✓ 5,9 ✓ 5,9 ✓ 5,2 ✓ 0 MA49A 10 MA44A 10 MA41B 10 MA42A 10 MA41A 10 MA46A 0 MA49A 10 MA44A 5,7 ✓ 10 MA34B 10 ME46A 4 Q130A 10 Q132A 8 Q133A 4,7 ✓ 5,2 ✓ 4,5 ✓ 4,9 ✓ 6,8 ✓ TOTAL 212 UD
OBLIGATORIOS: 100 UD.	PROMOC. LICENCIATURA:
10 CC 51A 10 CC 52A 10 CC 52B 00 CC 59A : 20 CC 61A 00 CC 65A 04 CC 65E 36 CC 69FICC 69H 10 IN 53A	100 UD
CARRERA: INGENIERIA CIVIL EN COMPUTACION	TERMINO LICENCIATURA:
(*) ELECTIVAS: 100 UD.	PROMOC. LICENCIATURA:
8	5) Se aceptarán hasta 12 UD en cursos libres, imputables a la cuota de electivos de la carrera.

GRADO: LICENCIATURA EN C.S. DE LA ING., MENCION COMPUTACION	FECHA:.....
NOMBRE:.....	AÑO INGRESO:.....
Nº MATRICULA:.....	RUT:.....
PLAN COMUN FACULTAD: 242 UD.	ELECTIVOS LIC.: 28 UD.
LICENCIATURA: 140 UD.	LICENCIATURA: 140 / 140 U.D.
20 CC 10A 20 FI 10A 09 FI 21A 09 FI 21B 09 FI 22A 05 FI 25A 09 FI 33A 09 FI 34A 05 FI 35A 20 MA 11A 20 MA 12A 09 MA 22A 09 MA 25A 09 MA 26B 09 MA 33A 09 MA 34A 05 SU 20A 09 QH 21A 00 IDIOMA 09 EH	10 CC 30A 10 CC 31A 10 CC 31B 10 CC 40A 10 CC 41A 10 CC 41B 10 CC 41C 10 CC 42A 10 IN 34/INMA 3/A 10 IN 41A 10 IN 42A 10 IN 46A 10 MA 34B 09 MA 33A 09 MA 34A 05 SU 20A 09 QH 21A 00 IDIOMA 09 EH
PLAN COMUN FACULTAD: 242 / 242 UD.	LICENCIATURA: 140 / 140 U.D.
10 CC10A 0 EH09A 9 EH22A 9 EH22E 20 FH10A 9 F121A 9 F121B 9 F122A 5 F125A 9 F133A 9 F134A 5 F135A 20 MA11A 10 MA12A 0 MA22A 0 MA25A 0 MA26B 0 MA33A 0 MA34A 0 SU20A 0 QH21A 0 IDIOMA 0 EH	5,5 ✓ 7 ✓ 5,4 ✓ 5,3 ✓ 4,6 ✓ 4 ✓ 5,8 ✓ 4,7 ✓ 5,9 ✓ 5,9 ✓ 5,2 ✓ 0 MA49A 10 MA44A 10 MA41B 10 MA42A 10 MA41A 10 MA46A 0 MA49A 10 MA44A 5,7 ✓ 10 MA34B 10 ME46A 4 Q130A 10 Q132A 8 Q133A 4,7 ✓ 5,2 ✓ 4,5 ✓ 4,9 ✓ 6,8 ✓ TOTAL 212 UD
OBLIGATORIOS: 100 UD.	PROMOC. LICENCIATURA:
10 CC 51A 10 CC 52A 10 CC 52B 00 CC 59A : 20 CC 61A 00 CC 65A 04 CC 65E 36 CC 69FICC 69H 10 IN 53A	100 UD
CARRERA: INGENIERIA CIVIL EN COMPUTACION	TERMINO LICENCIATURA:
(*) ELECTIVAS: 100 UD.	PROMOC. LICENCIATURA:

Figura 1.4: Planilla base ocupada para el desarrollo de la memoria descrita en [2]

En la figura 1.4 se puede observar gran parte del trabajo realizado en [2] ya que, tomando como base las planillas del Secretario de Estudios, se fue construyendo una plataforma similar de manera que se evitaran los errores humanos de tipeo. Aun así, la lógica aun estaba a cargo de una persona que debía corroborar la información y realizar cambios en la ubicación de los ramos hasta lograr una combinación factible.

Finalmente, el problema (que se abordará de manera más profunda posteriormente) consiste en asignar ramos dentro de planes de manera de encontrar una combinación que satisfaga las dos condiciones mencionadas en los antecedentes. Este tipo de problemas, de calcular múltiples combinaciones, puede tomar mucho tiempo. Aun así, dado el entendimiento del caso que se ha hecho y los intentos previos de automatización, se puede lograr una reducción ad-hoc que permita lograr el desafío y por ende motiva aún más la búsqueda.

1.3. Objetivos

Los objetivos de este trabajo de memoria se describen a continuación:

General:

- Rediseñar e implementar el actual sistema de recuento de UD's para asegurar la confiabilidad de los resultados, tanto desde el punto de vista del cumplimiento del plan de estudios como del cálculo del promedio de titulación, de manera tal que al ser utilizado se resuelva en un tiempo razonable para un sitio web (menos de 15 segundos).

Específicos:

1. Enunciar el problema, sus distintas extensiones y restricciones.
2. Buscar posibles algoritmos de solución en el estado del arte .
3. Analizar la potencial factibilidad para tomar una decisión respecto a un método.
4. Seleccionar un método para la verificación del cumplimiento del plan de estudios.
5. Seleccionar a lo sumo dos métodos para maximizar el promedio del alumno, de manera de acotar el desarrollo futuro.
6. Desarrollar el modelo y la implementación de los métodos seleccionados.
7. Establecer los casos tipo que sirven para validar la resolución del problema.
8. Evaluar los algoritmos con los casos tipo.

1.4. Metodología

El ADI posee un equipo de profesionales con distintas aptitudes que siguen una metodología. Inicialmente se realiza un estudio del lugar donde se encuentra la oportunidad de mejora, logrando un entendimiento profundo tanto de la situación actual como de las necesidades futuras. Luego, se replantean procesos y se diseñan las herramientas de apoyo de manera que,

finalmente, se implemente una solución ad-hoc.

Gran parte del entendimiento ya se posee, debido a las anteriores memorias y a la implementación previa que poseía el equipo. Por lo tanto, los pasos que se deben seguir serían el lograr entender individualmente el problema a partir de lo ya obtenido, comprender la implementación actual y rediseñar en conjunto con el equipo logrando un correcto modelamiento y pensamiento tanto del problema como la solución para, finalmente, implementar y validar las mejoras.

Las plataformas desarrolladas por el ADI poseen un ambiente de desarrollo. Esto quiere decir que se posee una copia de la herramienta, pero en un servidor distinto al público en donde pueden realizarse pruebas con datos reales o simulados. Además, están estructuradas en módulos administrables que permiten el manejo de permisos y aislación de manera tal que el desarrollo de una persona no afecte a otra.

U-Campus posee un sin fin de módulos instalados dentro del servidor en una carpeta compartida donde uno de ellos es el módulo principal. Para realizar una instalación local del sistema se debe poseer en el espacio propio una copia del módulo principal y la definición de las variables por defecto que mencionen dónde están las carpetas con los demás módulos. De esta forma, por defecto se cargarán todos los módulos desde la carpeta compartida y cuando se quiere realizar una mejora de alguno de los módulos se copia la carpeta necesaria en un espacio propio y se define una variable dentro de la instalación principal que fija el nuevo lugar de ese módulo. Aun así, para agrupar los cambios que se puedan hacer o las distintas versiones que puede tener el sistema se ocupa un controlador de versiones opensource conocido como “Subversion” (SVN).

Además, en cada módulo dentro de los sistemas se divide la capa lógica de la capa de presentación siguiendo lineamientos del paradigma MVC (modelo-vista-controlador). Es decir, dentro de cada módulo se pueden encontrar tres carpetas: templates, include y web. La primera sólo posee elementos con un lenguaje de marcado como HTML donde se definen mayoritariamente aspectos gráficos. En la carpeta include, mediante el lenguaje de programación PHP, se definen funciones que centralizan los modelos, los cuales realizan consultas a la base de datos o procesamientos amplios reutilizables. Finalmente, en la tercera carpeta (web) se poseen los controladores, es decir, es la carpeta pública con archivos que son llamados desde el exterior que orquestan las funciones que se deben llamar desde la carpeta include y determina el template (archivo HTML) con el cual se dispondrá la respuesta.

El recuento de UD's se encuentra en el módulo de “Boletines” (BIA) y gran parte de su lógica se encuentra en las funciones de la carpeta include. Por ende, la metodología a seguir para la implementación es crear una copia local de aquel módulo configurando las variables correspondientes y crear nuevas funciones que reemplacen a las anteriores sin mayor consecuencia a los demás elementos.

Finalmente, debido a que se poseen copias de los datos de alumnos, es posible verificar tanto la copia antigua como la nueva y chequear a su vez el buen desempeño de las funciones sin realizar pruebas unitarias. Esto debido a que la verificación de las mejoras y su comportamiento permitirán evaluar de una manera global todos los elementos partícipes del módulo.

Capítulo 2

Marco Teórico

Gran parte de los conceptos utilizados en esta memoria no son necesariamente de conocimiento general. Es por ello que a continuación se explicará de manera breve los elementos más utilizados.

2.1. Complejidad

Cuando se desean comparar procedimientos o algoritmos se debe utilizar alguna unidad de medida que permita definir cuál es mejor. Una unidad importante es el tiempo (cuánto tiempo demora uno con respecto al otro) pero éste es relativo a la cantidad de datos que se procesan, por ejemplo, es distinto intentar ordenar cien números a ordenar un millón. Por lo anterior, para comparar dos procedimientos, comúnmente se define una función $T(n)$ donde n es la variable que define el tamaño de los datos.

A partir de estos hechos y del procesamiento de los datos que se realiza en el algoritmo, se va describiendo el comportamiento de la función. Por ejemplo, si el algoritmo visita dos veces cada uno de los datos la función estaría definida por $T(n) = 2 \cdot n$.

Hay que tener en consideración, además, que esta definición es independiente de la tecnología utilizada para procesar. Es decir, si utilizamos una máquina con mayor capacidad se procesará una cantidad de datos determinada mucho más rápido, pero seguirá teniendo la misma función, por lo tanto es común enunciar la tecnología utilizada.

Usualmente no es utilizada directamente la función $T(n)$ y se usan asíntotas que determinan el mejor y peor caso en que se puede comportar el algoritmo. Esto permite un análisis más rápido debido a reducciones que se pueden utilizar. En esta memoria se trabajará con el peor caso que se denota con una letra O . Debido a la descripción de esta última función, se puede tener distintos tipos de complejidades, por ejemplo cuando se tiene una función $O(n^2)$ se dice que esta tiene un comportamiento cuadrático para hacer referencia al polinomio de segundo grado o cuando se tiene una función $O(3^n)$ se dice que es de orden exponencial debido a que la variable se encuentra en el exponente.

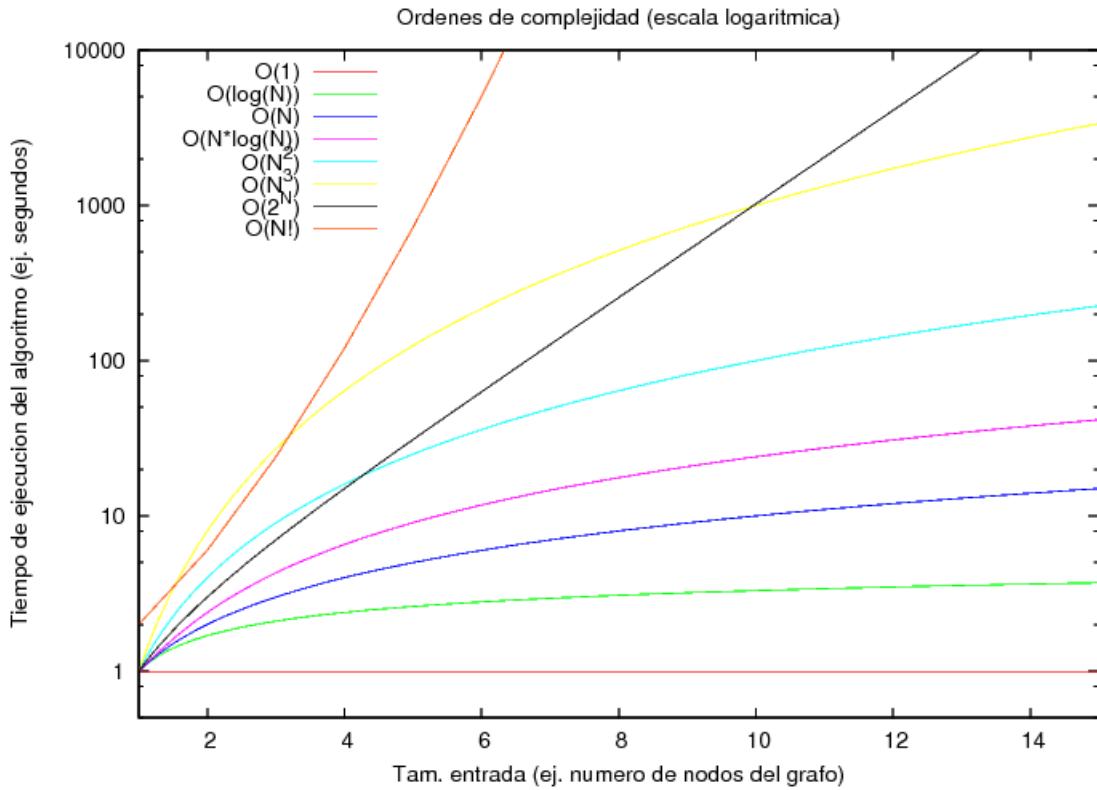


Figura 2.1: Gráfico con las curvas de órdenes de complejidad en una escala logarítmica

En la figura 2.1 se pueden ver los tiempos que demora en procesar un algoritmo una cantidad de datos determinada en una escala logarítmica, donde mientras más arriba diverge la función más difícil o más tiempo toma su cálculo. En computación se tienen distintos tipos de clases de complejidad para clasificar los problemas según la mejor solución descubierta por el momento, siendo las mas conocidas la clase P, los problemas que pueden ser resueltos en tiempo polinomial, y NP, los problemas que pueden ser resueltos en tiempo polinomial, pero sólo por una máquina no determinista. Estos últimos poseen un subconjunto de problemas conocidos con una complejidad mucho mayor y que son reducibles entre ellos, es decir, un problema puede ser representado o transformado a una variante equivalente de otro problema dentro de ese subconjunto.

Otra unidad de medida diferente al tiempo es la unidad de espacio. Con esto se evalúa la cantidad de memoria necesaria que requiere una solución. En esta tesis se tendrá énfasis en la unidad de tiempo, pero en la sección “Validación” se mencionará brevemente los cuidados que hay que tener con el espacio.

2.2. Teoría de Grafos

Un campo que ha tomado gran importancia en la computación es la teoría de grafos. Los grafos son una estructura que está compuesta por dos elementos: los nodos y las aristas. Las

aristas pueden tener dirección y en ese caso se habla de un grafo dirigido.

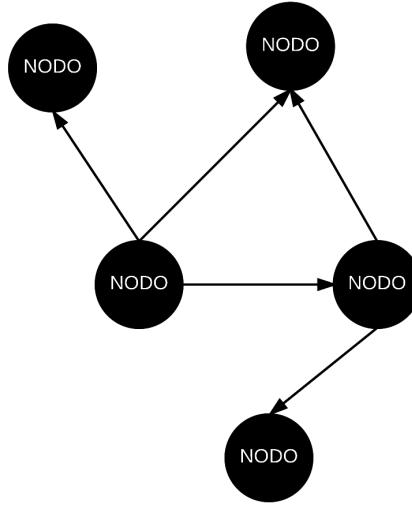


Figura 2.2: Ejemplo de grafo dirigido.

La forma que posee esta estructura ha sido de utilidad, debido a que muchas cosas pueden representarse mediante ella. Por ejemplo, posiciones dentro de un mapa pueden ser representadas por nodos y las aristas son los caminos entre ellas, teniendo la habilidad de agregar valores (la arista tiene un valor mayor mientras más alejados están los nodos). A partir de esto y una representación matemática de los elementos de un grafo pueden resolverse problemas teóricos con variada utilidad en la práctica. Siguiendo el mismo ejemplo anterior, la solución teórica del camino con aristas de menor valor entre dos nodos podría representar el camino más corto entre dos lugares geográficos.

Existen variados tipos de grafos, entre ellos los grafos conexos que se definen como los grafos donde para todo par de nodos se tiene un camino que los une. Cuando no se posee un grafo totalmente conexo se pueden tener subconjuntos que sí lo sean a los que llamaremos componentes conexas. En la imagen 2.3 se puede observar un grafo con componentes conexas separadas por las líneas punteadas. Si existiera un arco entre los nodos 7 y 6, y entre los nodos 2 y 10, se tendría un camino entre todos los nodos y, por lo tanto, un grafo conexo.

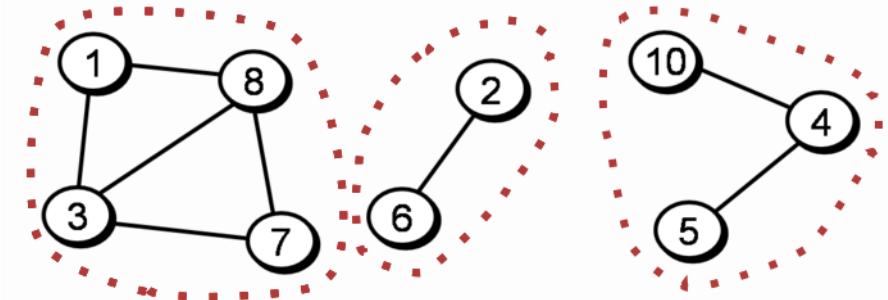


Figura 2.3: Ejemplo de componentes conexas.

Otro caso particular de grafo son los árboles. Este es un grafo conexo en que la unión de

todo par de vértices es única, es decir, se posee sólo un camino entre un nodo y otro. En este caso, si bien las aristas en un árbol no poseen dirección, se puede armar un estilo de jerarquía donde el nodo desde donde se originan las primeras aristas es conocido como la raíz. Además, es común en este tipo de estructuras describir los nodos como padres e hijos de acuerdo al nivel que poseen y desde cuál nodo se originan. Los nodos que ya no poseen nodos hijos son conocidos como las hojas del árbol.

Dentro de la teoría de grafos y las componentes conexas, existe un concepto útil de comprender para esta memoria conocido como “la componente gigante” (Giant component en inglés). Si bien es un estudio que implica variadas cosas y utiliza conceptos de probabilidades, uno de los elementos importantes es que si se posee un grafo de n nodos y se van adicionando aristas aleatorias, si la cantidad de aristas adicionadas supera aproximadamente $n/2$ con alta probabilidad se tendrá una componente conexa gigante.

2.3. Optimización Combinatorial

La optimización es un área de las matemáticas donde se busca determinar la mejor manera de realizar una actividad bajo criterios extraídos del modelamiento del problema. Un ejemplo es maximizar los ingresos de alguna institución o disminuir los tiempos de ejecución de una tarea.

En el caso particular, la optimización combinatorial involucra encontrar la solución dentro de un conjunto discreto. Comúnmente, en los problemas relacionados se tiene la opción de encontrar la solución con una búsqueda exhaustiva, pero debido a la magnitud del conjunto el tiempo necesario para llegar a la confirmación de que se está en presencia de la mejor solución se vuelve muy grande y por lo tanto el método no es factible.

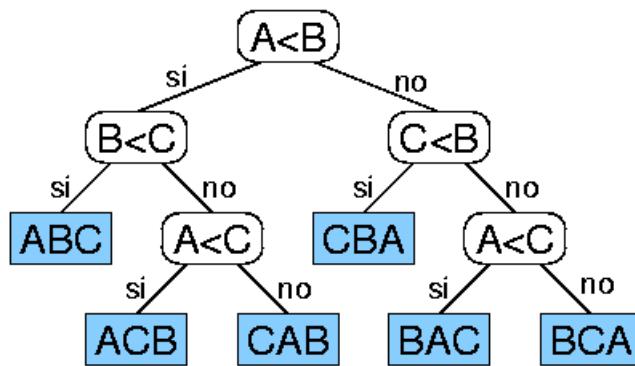


Figura 2.4: Árbol de decisión para ordenar tres elementos A, B y C.

Usualmente, un problema de optimización combinatorial puede representarse como un árbol de decisión. Es decir, un grafo donde cada nodo diverge hacia otros nodos según la cantidad de opciones que se tiene sobre una variable, de esta manera para cada variable que se debe definir se tendrá una representación visual de alguna elección. Ejemplificando, si tenemos un problema de dos preguntas que se pueden responder con sí o no, tendremos

un nodo donde se decida sobre la primera pregunta y se tendrá dos aristas debido a las dos opciones de respuesta. Luego, en los nodos hijos se representará la decisión de la segunda pregunta, pero teniendo como dependencia la decisión ya tomada, por lo tanto las hojas de este árbol de decisión representan las distintas combinaciones que se pueden obtener del problema conjunto.

Variados problemas de optimización combinatorial presentan una complejidad de clase NP y muchos de ellos pueden ser representados con grafos. Además, como no se logra encontrar una solución rápida, los problemas son estudiados en profundidad documentándose distintos avances con algoritmos y heurísticas. Estos pueden tener mejoras en casos particulares y es importante conocer algunos de ellos que se parezcan al caso de la presente memoria.

Casos conocidos son el problema de la mochila o el de mudanza. En ellos es necesario poner objetos dentro de algún contenedor de tal manera de maximizar la cantidad de objetos que se pueden llevar, maximizar la utilidad de esos objetos, minimizar la cantidad de contenedores necesarios u otra optimización. Martello y Toth [3, 4] propusieron algoritmos para resolver este tipo de problemas siendo uno de ellos la estrategia de ramificación donde el ordenamiento de los objetos se realiza por valor decreciente (tamaño, peso o importancia, dependiendo del objetivo). De esta manera, para cada nodo de decisión se tiene el objeto de mayor valor y es ubicado en el contenedor de la más próxima factibilidad.

Cuando no se posee una solución exacta adecuada se puede incurrir en reducciones del conjunto de combinaciones con relajaciones de las restricciones. Aun así se deben tener presentes conceptos como el de programación dinámica para no incurrir en cálculos reiterados de un subconjunto. Para exemplificar lo anterior podemos intentar resolver la sucesión de Fibonacci la cual esta dada por:

$$F(n) = F(n - 1) + F(n - 2), F(0) = 0, F(1) = 1$$

Para calcular el valor de $F(5)$, de acuerdo a la secuencia, se debe calcular el valor de $F(4)$ y $F(3)$. Aun así, para calcular el valor de $F(4)$ se debe calcular nuevamente el valor de $F(3)$ y el valor de $F(2)$. Si se siguiera el orden anterior y fuese efectuado por un computador, el cálculo de $F(3)$ se realizaría repetitivamente siendo que no es necesario si ya fue calculado una vez, por lo que se podrían realizar dos mejoras a la forma en que fue orientada la solución del problema: tener en memoria los cálculos ya hechos para consultarlos posteriormente o seguir una estrategia bottom up (comenzar desde $F(1)$ hasta llegar a $F(n)$).

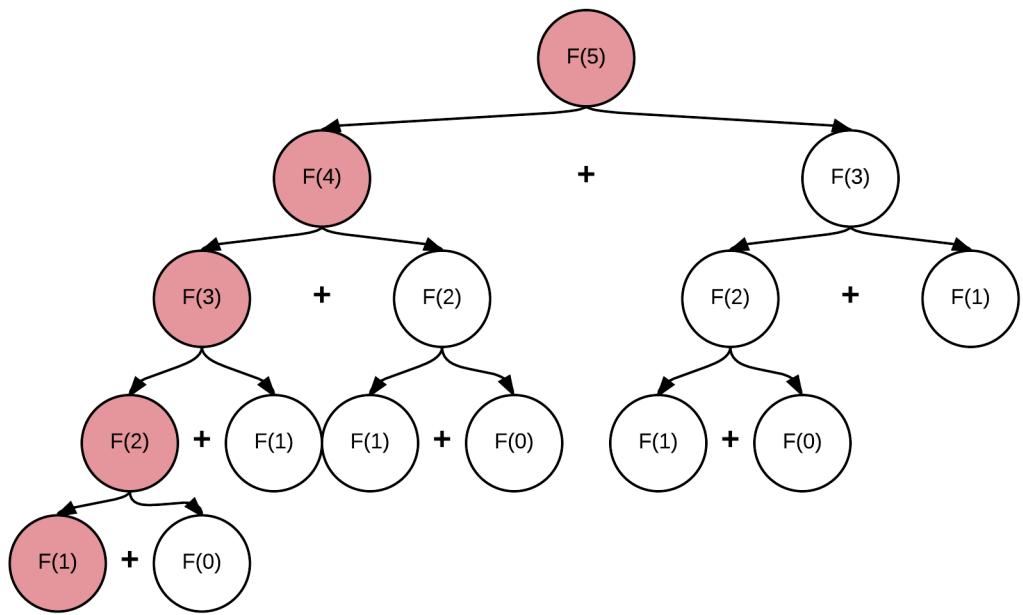


Figura 2.5: Cálculos hechos en serie de Fibonacci siguiendo perspectiva top-down. En rojo se puede observar los cálculos necesarios si se siguiera una perspectiva bottom-up.

Capítulo 3

El Problema: Situación Actual

El recuento de créditos inicialmente era un proceso netamente manual. Estaba constituido por un solo actor que a partir de la experiencia podía manejar las diferentes condiciones que se generaban. Ante eventualidades, como casos extraños en el que faltaban UD's en cierto plan, el mismo actor tenía la potestad de flexibilizar las reglas para chequear si realmente un alumno cumplía con la carrera.

Como se mencionó en la descripción del proyecto, este proceso fue mejorando poco a poco logrando que existieran componentes automáticas en el procedimiento, tratando de suplir todo el conocimiento generado por el actor mencionado en el párrafo anterior. Aun así, la parte automatizada posee errores, de tal manera que se debe realizar un chequeo manual posterior.

Los errores se deben a la complejidad de las reglas que se deben cumplir y a la naturaleza del problema. Por ende, se ahondará en ello en la sección “Combinaciones” explicando el modelo e implementación actual.

Finalmente, en la sección del proceso actual se abordará el procedimiento realizado por la SGD para suplir la falta de certeza de la heurística en ciertos casos y las estadísticas de ocurrencia de los errores.

3.1. Combinaciones

Para entender el problema de forma más detallada podemos realizar una simplificación del recuento y reducirlo a un problema de mudanza (bin packing). Para ello debemos asimilar los planes como si fueran cajas, y los ramos como si fueran elementos que deben ser guardados dentro de estas cajas para poder ser trasladados.

Para este problema la solución trivial sería ir intentando distintas combinaciones posibles, de tal manera de lograr que todos los elementos sean guardados. Esto sería la solución a fuerza bruta del problema entregando la mejor solución, pero el tiempo necesario para realizar todas

las combinaciones posibles es muy grande. No se posee un tiempo ilimitado y, por ende, lo más natural sería comenzar a privilegiar algunos elementos guardando inicialmente los de mayor valor para luego ir llenando con los de menor valor o los que se podrían dejar olvidados (como lo realizado por Martello y Toth descrito en el “Marco Teórico”).

En el caso de la solución presente del recuento de UD’s se utiliza la anterior heurística. Se consideran los ramos de valor aquellos que poseen mayor nota y son de nivel más alto (se debe privilegiar poner los ramos de mejor nota en la carrera debido a que ellos son considerados para el promedio final). Finalmente, esto iría mediante backtracking visitando todas las soluciones posibles, pero teniendo en un inicio ramos de importancia fijos.

Matemáticamente, se puede definir que un ramo r puede pertenecer a cierto conjunto de planes P_r de tamaño n_r , y que un plan se ve cubierto si y sólo si tiene asignado una cantidad determinada de ramos mayor o igual a L_p . Por lo anterior, el determinar el cumplimiento del plan y la nota se trata de un problema de asignación en el que se puede usar el ramo en alguno de sus planes. Si consideramos $X_{rp} \in \{0, 1\}$ como la variable de decisión que determina si se usa el ramo r en el plan p , el modelamiento del problema de calcular si se cumple el plan quedaría determinado por:

$$\begin{aligned} & \max \sum_r \sum_{\text{Ramos}}^{\text{Planes}} X_{rp} \\ \text{s.a. } & \sum_r^{\text{Ramos}} X_{rp} > L_p \quad \forall p \in \text{planes} \\ & \sum_p^{P_r} X_{rp} \leq 1 \quad \wedge \quad \sum_p^{\text{planes} \setminus P_r} X_{rp} \leq 0 \quad \forall r \in \text{ramos} \end{aligned}$$

De esta forma, el orden de complejidad de este modelo vendría dado por la pitatoria $\prod_r n_r$. Es decir, en el caso hipotético en que m ramos tuvieran tres opciones donde ser ubicados se tendría $\prod_r 3 \Rightarrow O(3^m)$, lo que equivale a un orden de complejidad exponencial. A partir de esto se descarta la posibilidad de un problema mucho mayor en que al ser combinaciones de todos los ramos contra todos los planes se obtendría un orden factorial de complejidad.

Como se posee poco tiempo en un sitio web, este módulo es detenido a los quince segundos y por lo tanto puede suceder que hasta ese tiempo no se haya visitado una combinación que diga que todos los planes se cumplen siendo que quizás más adelante podría encontrarla. En el caso del cálculo de la nota se sigue el mismo comportamiento debido a que se realiza en conjunto con la comprobación del cumplimiento del plan principal.

Si bien se asegura que se tiene una buena distribución porque se asignaron los ramos de importancia, no se tiene certeza de la exactitud en el punto donde es detenido el algoritmo. Aun así, se privilegió este procedimiento porque estos casos generan un falso negativo, es decir, el error que puede producir es que un alumno que puede titularse el sistema le diga que aún no puede. Para estos sucesos se siguen los pasos que se explicarán en la sección de “proceso presente”.

Aun así, lo descrito anteriormente sólo es una relajación del problema. Si se consideran todas las reglas específicas de cada plan se puede armar un modelo como sigue:

$$\begin{aligned}
 & \max \text{ } usar_{carr,carr} : carr \in planes \\
 \text{s.a. } & \sum_p^{\text{planes}} usar_{op} \leq 1 \quad \forall o \in \text{objetos} \setminus o \neq p \\
 & usar_{pp} \leq usar_{op} \quad \forall p \in \text{planes}, \forall o \in \text{objetos} \setminus \text{regla}_p = \text{todo} \wedge \text{pertenece}_{op} = 1 \\
 & usar_{pp} \cdot \text{cantidad}_p \leq contar_p \quad \forall p \in \text{planes} \setminus \text{regla}_p = \text{contar} \\
 & usar_{pp} \cdot \text{cantidad}_p \leq uds_p \quad \forall p \in \text{planes} \setminus \text{regla}_p = \text{ud} \\
 & usar_{pq} \leq contar_p \quad \forall p, q \in \text{planes} \setminus \text{regla}_p = \text{contar} \wedge \text{pertenece}_{pq} = 1 \\
 & usar_{pq} \leq uds_p \quad \forall p, q \in \text{planes} \setminus \text{regla}_p = \text{ud} \wedge \text{pertenece}_{pq} = 1 \\
 & usar_{pq} \leq usar_{pp} \quad \forall p, q \in \text{planes} \setminus \text{regla}_p = \text{todo} \wedge \text{pertenece}_{pq} = 1
 \end{aligned}$$

En este modelo se generaliza lo que son los ramos y los planes debido a que se comportan de manera parecida. Es decir, como un plan puede poseer tanto ramos como subplanes y además un ramo puede verse como un plan debido a que puede poseer equivalencias, finalmente pueden modelarse como un único conjunto llamado objetos (*ramos* \cup *planes*). Además, puede utilizarse una variable que en el modelo le llamamos “Usar” que implica que un elemento “i” está siendo usado en un elemento “j” de tal manera que esto implica los objetos considerados dentro de un plan. Un objeto estaría siendo usado en sí mismo si y sólo si es un ramo aprobado o es un plan donde se cubren todas las reglas relativas a él, y de esta manera se podría verificar cuando una carrera se cumple.

De esta manera las restricciones vendrían a explicar reglas parecidas al modelo simplificado como que un objeto puede ser usado sólo una ocasión, que cuando se tiene una regla “todo” se deben usar todos los elementos dentro de él y que cuando se tiene regla “contar” o “uds” se debe utilizar una cantidad mayor o igual a la determinada.

La forma en que está implementado actualmente el sistema es mediante backtracking. Es decir, lo que se realiza es tomar un ramo de una lista, fijar el plan donde puede ser utilizado (escogiendo el primero de los candidatos para ese ramo) y continuar con el ramo siguiente. En cada paso se va evaluando si la solución que se tiene en ese instante es mejor que una guardada en una variable global y si lo fuese se actualiza esa variable. Luego, cuando todos los ramos se asignaron, el algoritmo debe ir devolviéndose de manera tal de ir probando los siguientes candidatos de cada ramo, pero volviendo a calcular las combinaciones de los ramos siguientes porque cambió la situación.

Ejemplificando, si se poseen tres ramos A, B y C, se deben asignar los dos primeros y calcular todas las opciones para C. Luego, se debe retroceder un paso, cambiar el candidato de B y volver a calcular las opciones de C. Así, de manera sucesiva hasta lograr calcular todas las opciones de B y A, respectivamente. Como se dijo anteriormente, no se posee un tiempo infinito y, por ende, el algoritmo debe detenerse, por lo tanto es probable que no se logre retroceder hasta A y sólo se quede con el cálculo de las mejores opciones de B y C.

Es por ello que los ramos determinados en un principio deben asignarse de la mejor manera posible para que el costo de dejar el algoritmo inconcluso se minimice.

3.2. Proceso Presente

Inicialmente, gran parte del recuento de UD's pertenecía a “Secretaría de Estudios” ya que ellos verificaban el cumplimiento de los requisitos académicos del proceso de titulación. Aun así, por la complejidad del proceso y las fallas encontradas en el módulo automático la Subdirección de Gestión Docente (SGD) debió ser parte del proceso para generar una propuesta inicial.

Este proceso inicia con la aprobación de un ramo terminal. Es decir, al cumplir un ramo que está cercano al término de la carrera, se puede confirmar que un alumno va a necesitar la comprobación de sus antecedentes académicos. Inicialmente es necesario procesar la licenciatura asociada a la carrera del alumno para proseguir normalmente.

Proceso	Estado	Responsable
Requisitos Académicos		
<input type="checkbox"/> Tramitar Licenciatura Comprobar que exista una Licenciatura asociada a la Carrera y que tenga recuento de UD's completo	Recuento de UD's de Licenciatura Incompleto	Subdirección de Gestión Docente
<input type="checkbox"/> Aprobar Ramo Terminal Los ramos terminales corresponden a aquellos ramos que permiten completar el plan de estudios	Requisitos: Tramitar Licenciatura	Automático
<input type="checkbox"/> Propuesta de Recuento de UD's - Pregrado Subdirección de Gestión Docente debe enviar una propuesta del recuento de UD's	Requisitos: Tramitar Licenciatura	Subdirección de Gestión Docente
<input type="checkbox"/> Recuento de UD's Comprobar que el alumno cumple con los cursos necesarios para el programa	Requisitos: Tramitar Licenciatura	Secretaría de Estudios

Figura 3.1: Imagen del módulo de titulación donde se muestran los procesos necesarios para comprobar los requisitos académicos.

Luego, la SGD valida el resultado entregado por el módulo del recuento de UD's verificando que no se esté en presencia de un falso negativo. Esto se evidencia cuando el sistema menciona que el alumno no cumple el plan y existen ramos no usados que con un re-ordenamiento de los ramos utilizados pueden disponerse y lograr una combinación válida.

La búsqueda de una combinación puede ser lenta y por ello se pueden recibir notificaciones de parte del alumno que busca que su proceso continúe como es debido. Además, él posee mucho mayor conocimiento de su avance curricular y, por ende, puede ayudar a la búsqueda de la solución. Dado esto, existe un módulo en U-Campus que permite la interacción entre alumnos e integrantes de la SGD.

Se posee un promedio de dos consultas por día relacionadas con el recuento. Estas son resueltas comúnmente en algunos días pero existen ocasiones en que toman mayor tiempo (meses) debido a que involucran normas adicionadas en los cambios de planes de estudios.

Estadísticas

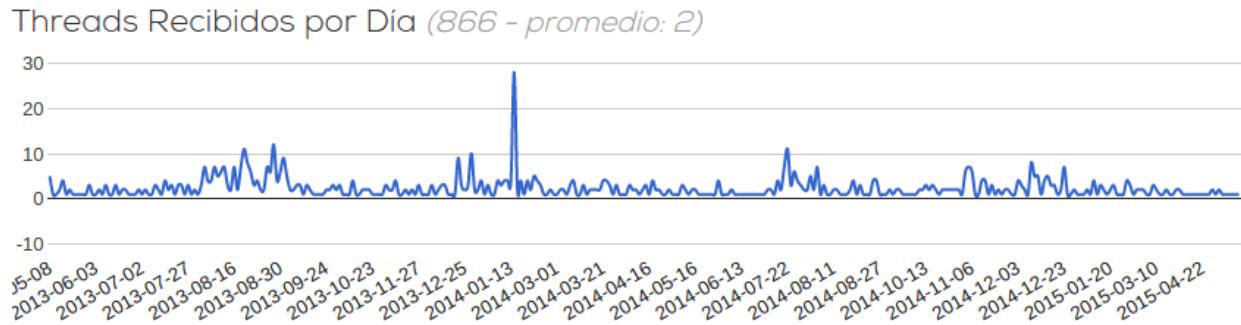


Figura 3.2: Estadísticas de peticiones por día relacionadas al recuento de UD's.

Esto está íntimamente relacionado con las versiones de los planes explicadas en la descripción del proyecto.

Finalmente, cuando ya se posee una combinación válida o cuando el alumno aprueba lo necesario para el cumplimiento del plan, se envía el resultado del sistema con observaciones de los cambios necesarios para obtener la combinación válida si se requiere. Ésta es recibida por Secretaría de Estudios quien valida la propuesta donde la acepta, confirmando el cumplimiento de los requisitos académicos, o la rechaza, para que la SGD justifique de mejor manera las observaciones agregadas.

Otros de los procesos que tiene a cargo la SGD son la inscripción académica, la construcción del catálogo de cursos, el calendario académico, entre otros. Muchos de ellos presentan alta carga y con tasas de errores dependientes del tiempo involucrado en ellos para una correcta planificación. Si se posee una mejora en la certeza del recuento de UD's se disminuirían los reclamos por parte del alumnado y la cantidad de rechazos por parte de Secretaría de Estudios.

Capítulo 4

Solución: Modelo y Rediseño

Dentro de las opciones mencionadas en un problema de optimización combinatorial se encuentra la reducción del espacio de soluciones. Es decir, un buen entendimiento y modelamiento del problema deberían implicar un espacio mínimo, pero no necesariamente, y es por ello que en la implementación anterior aún pueden existir mejoras orientadas a ello.

En un recuento de UD's un alumno que está cercano a titularse posee aproximadamente 70 ramos. Un plan de título profesional posee alrededor de 18 subplanes, los cuales poseen distintas reglas y por ende los ramos antes descritos pueden utilizarse o no, dependiendo estas normas. Aun así, pensando en la flexibilización inicial que se hizo en la sección "Problema", si los ramos poseen en promedio tres lugares donde ser ocupados tendríamos un número cercano a 3^{70} combinaciones que recorrer para decidir cuál sería la solución correcta. Si se eliminara un ramo se tendría tres veces menos combinaciones.

En la imagen 4.1 se observan las reglas de los planes de la tercera versión de la carrera de Ingeniería Civil en Computación. Han sido dispuestos como un árbol de acuerdo a la dependencia que tienen entre ellos ya que, recordando, un plan puede tener subplanes dentro de él. Si desde la raíz del árbol se va bajando de nivel se puede observar que se sigue una línea donde la regla de los planes es "todo", es decir, se deben cumplir obligatoriamente todos los elementos pertenecientes a esa línea. Por ende, todos los ramos que se encuentran dentro de esos planes podrían verificarse en un inicio reduciendo la cantidad de ramos que pasarían posteriormente por el algoritmo.

Si bien se puede pensar que esta es una reducción considerable, ya que se puede sacar del cálculo cerca del 50% de los ramos, de todas formas no es del todo significativa, debido a que comúnmente estos ramos son considerados obligatorios dentro del modelamiento de los planes y por ende poseen a lo más dos lugares donde ser ocupados. Además, se debe tener precaución debido a que, por ejemplo, los ramos dentro del plan de regla "todo" que está en el nivel inferior poseen un plan padre que tiene regla "contar" y podría cumplirse sin la necesidad de aprobar los ramos del subplan.

Siguiendo la misma línea pueden reducirse los ramos que poseen un lugar donde ser ocupados debido a que si se agregaran al procesamiento del algoritmo, éste considerará que pueden

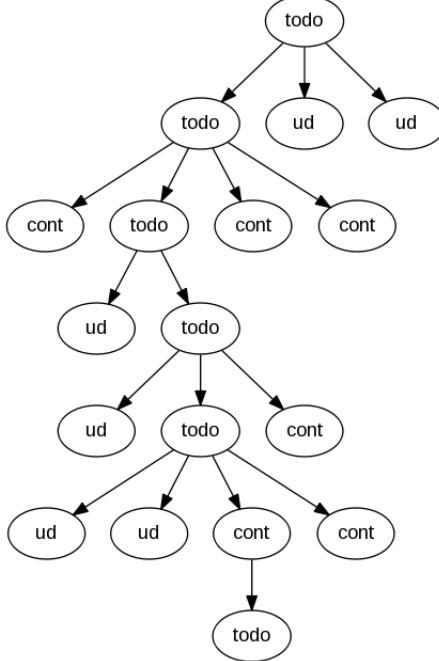


Figura 4.1: Árbol de planes con sus respectivas reglas como etiqueta de los nodos.

ocuparse o dejarse fuera del cálculo, y por ende no se estaría en presencia de tan solo un lugar donde ser ocupado, si no que dos. Eso si existe un diferencia entre este tipo de ramos que con los con planes con regla “todo”, ya que estos realmente podrían no ocuparse. Aun así, esta reducción podría considerarse dentro de las opciones porque para aprobar una carrera se podría ocupar la máxima cantidad de ramos posibles siendo perjudicada la nota, pero se realizaría un post-procesamiento para mejorar aquello.

La falencia que se puede apreciar en la implementación actual es que realiza una reducción descartando poner un ramo dentro de un plan si éste ya está cubierto. Si se realizarara la reducción del párrafo anterior, se agregarían ramos con menor nota y el orden de poner los ramos de mejor nota inicialmente se vería entorpecido, provocando que la reducción de la implementación actual funcione incorrectamente. De todas formas se implementarán ambas y se realizará una comparación para decidir cuál es mejor dentro de ellas, pero, a priori, debido a que los ramos reducidos de la implementación actual pueden tener más de una opción donde ser ubicados, implicaría una reducción mayor y sería una mejor decisión.

Yendo a la búsqueda de mejora dentro de la heurística ya implementada, se debe encontrar una perspectiva distinta para ver el problema. Si se modelara como un grafo podríamos encontrar formas dentro de la teoría que podrían servir como mejora. Por ende, si se consideran los ramos como si fueran nodos unidos sólo si poseen un plan en común dentro de sus opciones, se tendría una imagen como la de la figura 4.2.

Se observa un grupo de ramos de un alumno de la carrera de Ingeniería Civil en Computación, en donde el nodo posee el nombre del ramo en su interior y entre ellos están unidos por aristas de distintos colores. El color indica el plan por el cual están unidos existiendo ramos donde se origina un solo color debido a que sólo tienen un candidato donde ser ubicados y

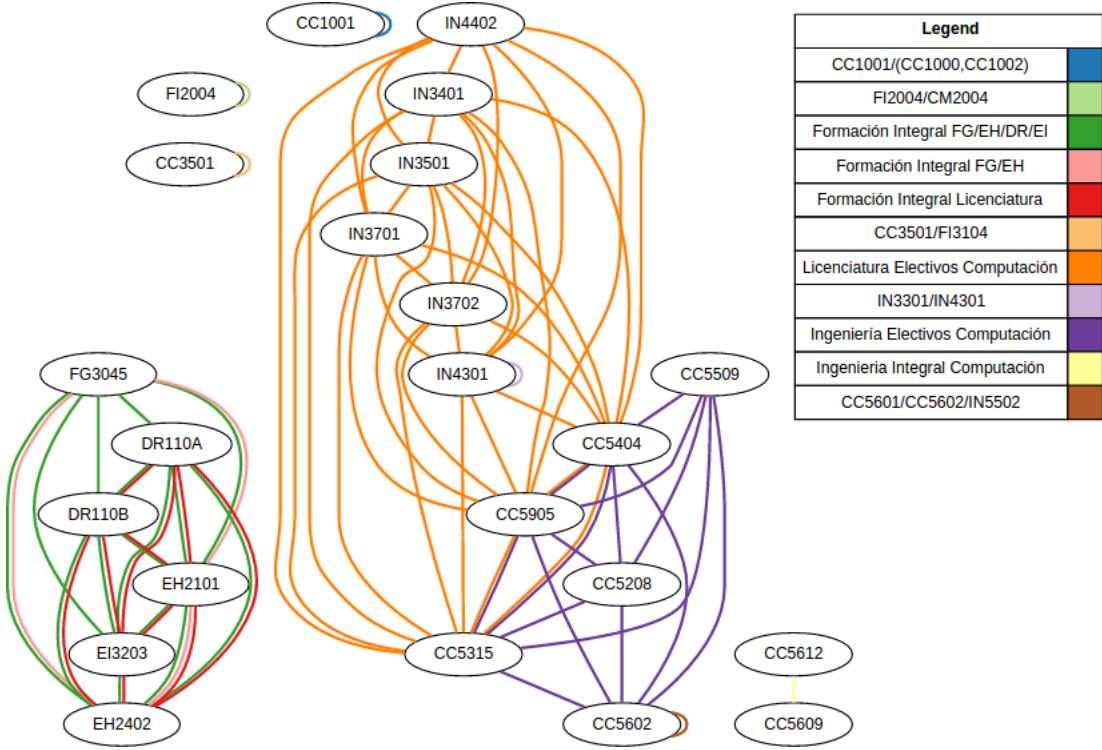


Figura 4.2: Conversión del problema en un grafo.

pueden no estar asociados a ningún otro nodo.

Además, como se observa, existen componentes conexas dentro del grafo. Si bien esto a priori puede no significar mucho, si se contrasta con la implementación actual toma sentido un tipo de mejora, como la programación dinámica antes explicada. Si se toma un ramo A y se puede gastar en los planes P o Q, sea cual sea la decisión con ese ramo, esto no va a afectar la decisión de un ramo B que se puede gastar en los planes R o S. Esto debido a que no comparten ningún plan y pueden tomarse como problemas independientes. Se volverían dependientes uno sobre el otro si se tuviera un ramo C que pueda gastarse en Q o R (ambas opciones de A y B) debido a que si se gasta en Q implicaría menos espacio para A o si se gasta en R significaría menos espacio de solución para B. La solución actual está considerando todos los ramos y sus decisiones dependientes de los demás, siendo que lo que significan las componentes conexas dentro del grafo es que existen subproblemas independientes.

De todas manera, inicialmente se consideró una dependencia completa debido a la forma de árbol de dependencia de los planes. Por lo mismo, se deben aún considerar reglas que no están siendo visualizadas dentro del grafo, pero se ven en la visualización de reglas (figura 4.1) como lo es la posibilidad de no cumplir un plan debido a que el plan padre se cumple.

En el caso de la imagen 4.3, dado el cálculo de un grafo de ramos de la segunda versión de la carrera Ingeniería Civil Electricista se pintaron en el árbol de reglas de los planes dos componentes conexas (azul y rojo). Si se sigue el razonamiento de uno de los párrafos anteriores, los árboles de decisión de estas componentes conexas deberían estar separados, pero dado que un plan no necesariamente debe cumplirse, ya que puede cumplirse el padre, estas deberían estar juntas. Podemos aprobar los planes azules y sólo un plan rojo (el que

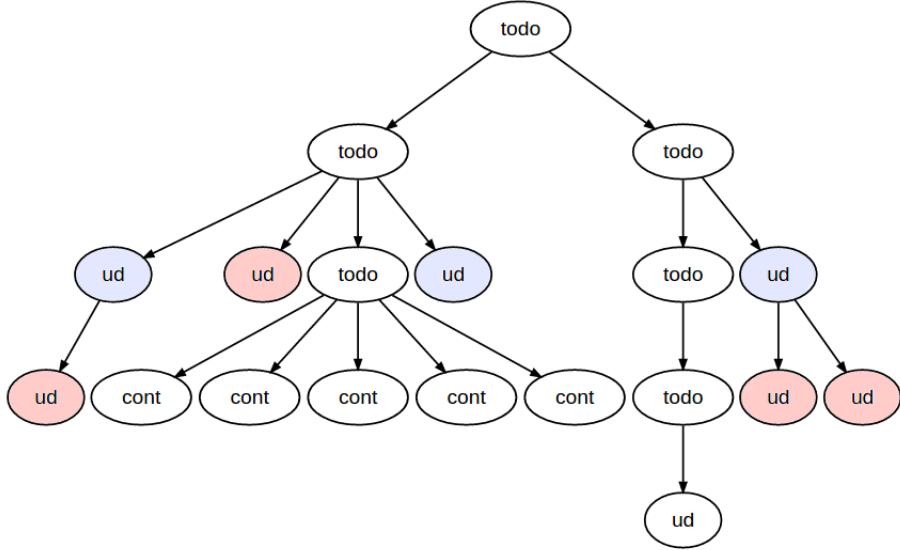


Figura 4.3: Árbol de reglas para la carrera Ingeniería Civil Electricista (v2).

no tiene un parente azul) y estaríamos cumpliendo todas las reglas, pero si lo consideramos problemas independientes en la componente conexa roja se hará esfuerzo por completar los planes de nivel bajo aunque no sea necesario.

Teniendo todas estas cosas en consideración, se lograría reducir el problema en varios problemas pequeños. Si anteriormente se tenían 3^{30} combinaciones, con esta mejora se lograría una reducción a problemas independientes y el cálculo se haría con sumas, es decir, si se pudiera dividir en tres componentes conexas con diez ramos cada uno se tendría la adición $3^{10} + 3^{10} + 3^{10}$ lo que es igual a 3^{11} combinaciones.

Si se vuelve a ver la imagen donde está el grafo con las componentes conexas (figuras 4.2 y 4.4), se puede ver que la componente con mayor cantidad de ramos posee aristas de color naranja y morado. Los ramos que están uniendo estos dos colores son tres (CC5315, CC5905 y CC5404) y si se toma inicialmente la decisión sobre ellos podrían formarse dos componentes conexas separadas. Es decir, decidir sobre un ramo podría formar nuevas componentes reduciendo aún más el espacio de soluciones. Aun así, el cálculo de qué ramos tendrían mayor probabilidad de ocasionar esta separación y el ordenamiento según ello podría significar un coste mayor, debido a que se estaría rompiendo el orden inicial de mayor valor (al igual que una de las reducciones que se pensó). Por lo tanto, se puede implementar el cálculo recursivo de componentes conexas y evaluar los dos posibles ordenamientos para ver cuál ocasiona una reducción mayor.

Finalmente, una última mejora a considerar en este trabajo es el caché. Este consiste en poseer parcialmente en memoria elementos, de manera que sea más fácil obtenerlos desde esta memoria que ser calculados o extraídos de otro lugar. Si bien con el cálculo de componentes conexas se estaría eliminando gran parte de los elementos repetidos, también pueden existir cálculos repetidos por la formación de nuevas componentes conexas que ya se habían formado anteriormente, las cuales pueden guardarse en memoria caché para no realizar su cálculo nuevamente.

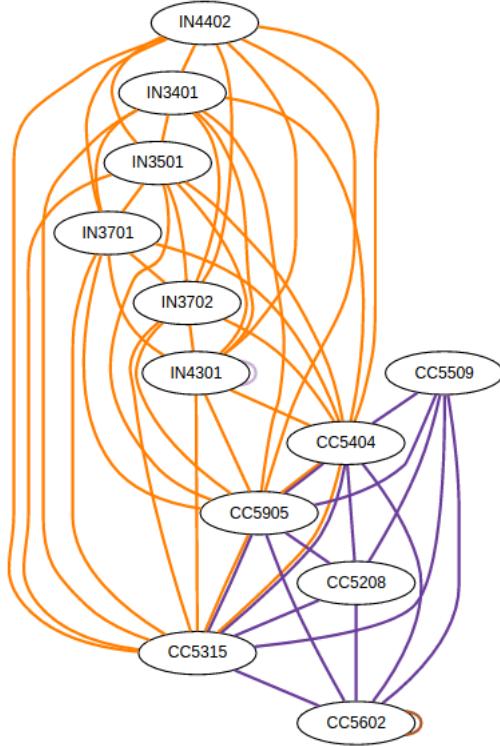


Figura 4.4: Extracto de la figura 4.2 para evidenciar la recursividad de las componentes.

Si se tuviera un ramo que al asignarse genera tres componentes conexas, el ramo podría haberse asignado dentro de esas tres opciones solamente. Si evaluamos paso a paso cómo se comportaría el algoritmo al asignar este ramo, podremos evidenciar el cálculo repetido.

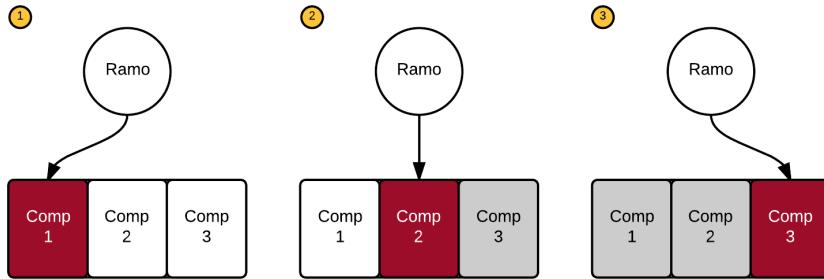


Figura 4.5: Comportamiento del caché. En rojo el lugar donde se ubica el ramo y en blanco la componente donde no se posee. En gris los cálculos que pueden ser obtenidos desde la memoria caché ya que ya fueron calculados anteriormente.

Al inicio, el ramo será asignado en la primera componente calculándola considerando el ramo dentro de su solución. En cambio las otras dos componentes se calcularán sin considerar el ramo. Una vez hechos esos cálculos, el algoritmo debe devolverse y calcular qué pasaría si el ramo fuera asignado en la segunda componente conexa. Allí, la segunda componente tendrá probablemente una solución distinta porque en esta ocasión debe considerar el ramo,

la primera componente también deberá hacer su cálculo nuevamente porque ahora no debe considerar el ramo (anteriormente sí lo había hecho), pero la tercera componente volverá a calcularse sin considerar el ramo, lo que podría guardarse dentro de un caché y no necesitar calcularla.

Lo anterior, debido al cálculo del algoritmo, sólo se produciría con ramos que generan más de dos componentes conexas lo que es de baja probabilidad. Aun así, se implementará dentro de esta memoria porque puede significar mejoras considerables si la componente conexa guardada fuese de un volumen grande.

Capítulo 5

Implementación

A continuación se mencionarán los pasos seguidos en la implementación. Estos serán explicados de forma concisa sin entrar en detalles de programación para un entendimiento más bien superficial de lo técnico, manteniendo una mirada global del cumplimiento del diseño y el modelo. De todas formas, en el primer apéndice se deja a disposición parte del código implementado para evidenciar detalles de la puesta en funcionamiento.

Para la implementación se utilizó como base lo hecho en las versiones pasadas del recuento de unidades docentes. En éstas el recuento se dividía en cuatro funciones importantes destacando dos de ellas: la inicial y la recursiva. La función inicial realiza la petición de los ramos y planes a partir del rut y carrera de un alumno, limpia estos datos, realiza asociaciones como las relaciones entre ramos de distintas versiones, llama a la función recursiva y, finalmente, retorna los resultados. Por otra parte, la recursiva realiza todo el procesamiento de la heurística anteriormente descrita llamándose a sí misma por cada asignación de ramo. Además, debe convocar a las otras dos funciones donde se propaga la elección de un ramo en un plan sobre todos los demás planes y el método en donde se evalúa si la solución actual es mejor que la guardada. Esta última también posee el contador de tiempo, donde si se exceden los quince segundos retorna la solución guardada por el momento y hace retornar los resultados desde la recursiva hacia la función inicial, para que esta los devuelva al controlador web que los solicitó.

En este caso, se utilizó la misma forma de programación procurando que todas las funciones tuvieran una firma similar entre ellas, respetando que siempre reciban la lista de ramos y planes, y retorne la mejor solución si corresponde. Además, como no estaba dentro del alcance, se reutilizó la implementación hecha en la función inicial de la implementación anterior. Por lo tanto, se poseen seis funciones, donde tres de ellas se corresponden con las explicadas en el párrafo anterior exceptuando la recursiva, debido a que se cambió el comportamiento del algoritmo.

Una de las funciones es el cálculo de las componentes conexas, donde a partir de los ramos se dividen en distintos arreglos debido a los planes que comparten. A esta función siempre se le entrega la lista de todos los ramos y planes, pero puede adicionarse una tercera variable opcional que dice que subconjunto de esas listas deben ser considerados para el cálculo y

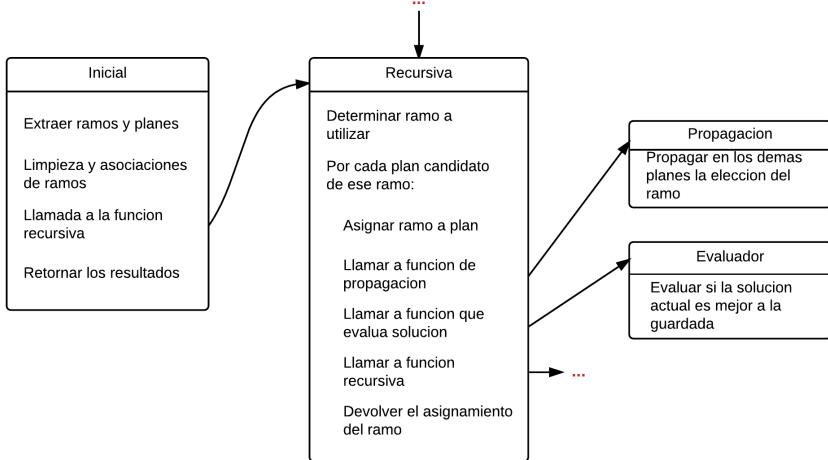


Figura 5.1: Funciones de la implementación anterior y sus comportamientos.

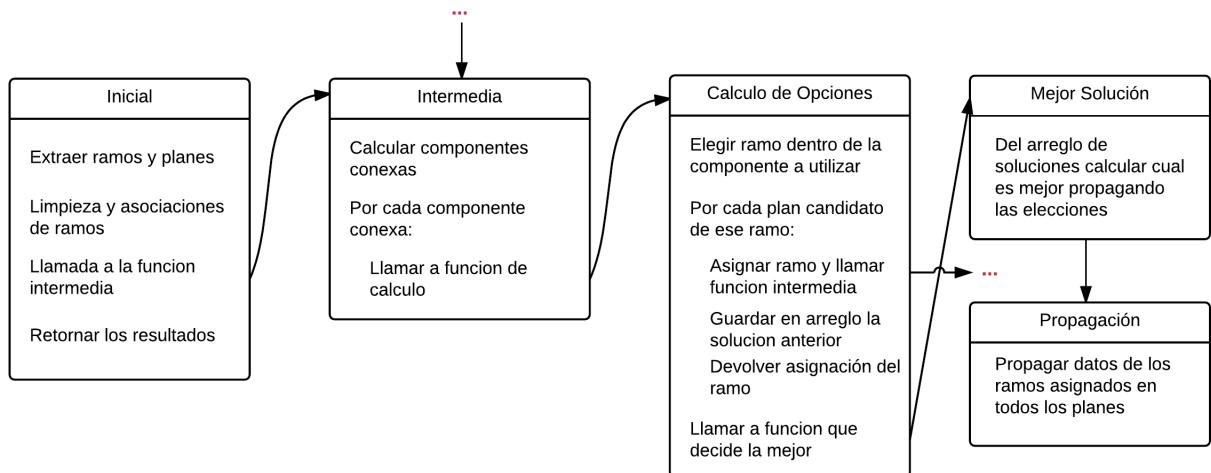


Figura 5.2: Funciones de la nueva implementación y sus comportamientos.

cuáles elementos deben quedar fuera.

Otra función es el cálculo de la mejor solución a partir de un arreglo de soluciones. En ella se recibe un arreglo y en una variable auxiliar se va guardando la mejor mientras se recorre el arreglo. Al término retorna aquella variable.

La función recursiva de la implementación antigua se cambió por dos funciones nuevas. La primera es el paso intermedio donde se calculan las componentes conexas y por cada una de ellas se inicia el cómputo de las distintas opciones de un ramo. La segunda es la que realiza este cálculo, elige un ramo y evalúa todas sus opciones (llamando a la primera función) generando un arreglo de posibles soluciones el que es pasado al método explicado en el párrafo anterior donde se calcula cuál es mejor.

En la primera función descrita en el párrafo anterior es donde debe ir ubicada la implementación del caché. Allí, se debe encontrar una firma (hash) que diferencie los elementos a guardar en memoria de manera tal de discriminar cuál componente conexa generada está repitiendo su cálculo para ser extraída desde la memoria y no necesitar el llamado a la segunda función.

Finalmente, se posee la función que realiza un recorrido con orden posterior (post order traversal) del árbol de planes para propagar las elecciones hechas en cada paso del segundo método explicado en el párrafo anterior.

Capítulo 6

Validación

Para validar la mejora en los tiempos y resultados efectuados se debe contrastar la ejecución del recuento de UD's para un grupo de alumnos con sus respectivas carreras inscritas. Para ello se poseen los datos de todos los estudiantes egresados y activos al año 2015.

Aun así, debido a que cuando se implementó la versión anterior y la carga de datos paulatina que se ha hecho, existen datos incompletos para una parte de los alumnos antiguos. Es por ello que, para chequear el real cumplimiento de todos los elementos hablados en esta memoria, se ocuparán solo los alumnos inscritos en las versiones más actualizadas de cada carrera.

Lo anterior implica el cálculo del recuento de unidades docentes de aproximadamente 4300 alumnos divididos en 12 planes distintos. Para este cálculo se realizó un script que desde la base datos obtuviera todos los alumnos anteriormente descritos para luego, dividiéndolos por carrera, ejecutar el algoritmo. Finalmente, se creó un archivo con el resultado de cada una de las carreras.

Código	Nombre
6	Geología
10	Ingeniería Civil, Mención Estructuras y Construcción
11	Ingeniería Civil, Mención Transporte
14	Ingeniería Civil, Mención Ingeniería Hidráulica, Sanitaria y Ambiental
15	Ingeniería Civil Eléctrica
16	Ingeniería Civil Industrial
17	Ingeniería Civil Matemática
18	Ingeniería Civil Mecánica
22	Ingeniería Civil de Minas
25	Ingeniería Civil Química
41	Ingeniería Civil en Computación
124	Ingeniería Civil en Biotecnología

Tabla 6.1: Tabla con los códigos de carrera y sus respectivos nombres.

El procedimiento se realizó tanto para el algoritmo antiguo como para el nuevo. De esta manera, es posible contrastar ambos resultados mediante las variables de tiempo con sus respectivos promedios y mediana, y a través de la cantidad de casos con tiempo mayor a 15 segundos (los casos que pueden ocasionar falsos negativos). Hay que tener en consideración en la ejecución de todos los recuentos que si bien tiene un tiempo prolongado, también realiza un uso exhaustivo de memoria y, por ende, se debía limpiar cualquier fuga que pudiese existir entre cada uno de ellos de manera de permitir la normal ejecución sin saturar el servidor. Además, se descartaron las opciones de realizar una asignación inicial de los ramos con un candidato, debido a que provocó desde un inicio resultados de mayor tiempo confirmando que ordenar los ramos de valor decreciente es la mejor opción, y la alternativa del caché no fue suficiente porque se veía perjudicado por el cálculo del elemento identificador (hash) de cada estado.

Carrera	Cantidad	Promedio	Elems. sobre 15s	Min	Max	Mediana
11	32	0,21984	0	0,01	1,243	0,075
124	94	2,3742	7	0,005	-	0,6965
14	175	0,3807	0	0,004	5,668	0,108
17	177	4,38622	38	0,001	-	0,223
25	191	5,12307	46	0,001	-	0,174
41	340	2,72421	33	0,001	-	0,0535
10	381	1,0309	2	0,006	-	0,037
22	390	0,82603	0	0,001	12,485	0,05
18	414	1,00994	8	0,001	-	0,337
15	484	2,6779	43	0,001	-	0,2
6	530	1,59584	16	0,001	-	0,141
16	1091	4,95989	297	0,001	-	15,007

Tabla 6.2: Tabla de resultados de la heurística antigua.

En la tabla 6.2 se presentan los resultados de la ejecución de la heurística antigua dividida por código de carrera. En los casos donde no se adiciona el tiempo máximo de ejecución es debido a que existen elementos sobre los 15 segundos y, por lo tanto, no se posee un registro de aquel tiempo. Además, hay que considerar que los promedios están reduciendo a 15 segundos los casos en que ese tiempo es mucho mayor, y por ende se ven reducidas las medias. Aun así se agregó el cálculo de la mediana para evidenciar la tendencia del algoritmo en todos los casos.

En cambio, en la tabla 6.3, se entregan los resultados para la ejecución del algoritmo nuevo. Si bien en estos casos se poseía un valor máximo para todos los casos, se decidió normalizar los valores mayores a 15 segundos al igual que la heurística antigua debido a la necesidad de que los promedios fueran comparables.

Las carreras fueron ordenadas por la cantidad de alumnos de manera creciente para ambos casos. Al comparar ambas tablas se puede observar un gran avance con la implementación nueva reduciendo en 9 veces los promedios de tiempo en la mayoría de los casos, a excepción de la carrera Ingeniería Civil Industrial (código 16).

Para poder explicar el comportamiento de la carrera Ingeniería Civil Industrial debemos

Carrera	Cantidad	Promedio	Elems. sobre 15s	Min	Max	Mediana
11	32	0,03863	0	0,008	0,135	0,0205
124	94	0,28724	0	0,005	3,838	0,1125
14	175	0,05555	0	0,003	0,759	0,022
17	177	2,38323	8	0,001	-	0,076
25	191	2,26128	7	0,001	-	0,076
41	340	0,36677	0	0,001	7,459	0,0315
10	381	0,1577	1	0,003	-	0,011
22	390	0,05714	0	0,001	0,39	0,0155
18	414	0,23142	0	0,001		0,12
15	484	1,09025	7	0,001	-	0,085
6	530	0,30912	1	0,001	-	0,058
16	1091	3,94389	201	0,001	-	4,197

Tabla 6.3: Tabla de resultados del algoritmo nuevo.

acudir al grafo realizado en la sección “Solución”, pero para el caso de este plan. En la figura 6.1 se puede observar cómo se generan componentes conexas correctamente como el nuevo algoritmo predijo, aun así, si se compara con el grafo de la figura 4.2, se pueden evidenciar dos grandes diferencias. Primero, las conexiones de la componente conexa más grande está realizada por ramos que comparten todos los planes. Es decir, no existe diferenciación o discriminación entre los dos planes que están en esa componente. Estos planes son “Licenciatura Electivos Industrial” e “Ingeniería Civil Industrial v3, Electivos”, y si bien no necesariamente implica un mal modelamiento de los planes de los ramos, ya que es una opción viable dentro de todas las alternativas, como opinión personal se considera que pierde el sentido diferenciar en dos planes distintos algo que en la práctica no se diferencia perdiendo la oportunidad de, por ejemplo, aprovechar realizar planes con distintas profundidades académicas.

Por otra parte, la segunda razón por la cual el comportamiento en esa carrera es más complejo para el algoritmo tiene que ver con el tamaño de la componente conexa mayor, donde se observa cerca del doble de ramos complejizando el orden desde 3^{10} a 3^{20} . La causa de esto es la cantidad de Unidades Docentes que poseen los ramos electivos en Industrias, ya que poseen tan solo 5 UD's, siendo lo normal de un ramo 10 UD's. Esto se debe a que sus ramos o poseen menor profundidad o son dictados con la misma exigencia que un ramo común, pero en mitades de semestre construyéndose periodos ficticios. Es decir, si bien el período académico de un semestre normal implica 15 semanas, estos cursos son dictados sólo por 7 semanas, ya sea desde el comienzo o iniciando retrasadamente desde la octava semana.

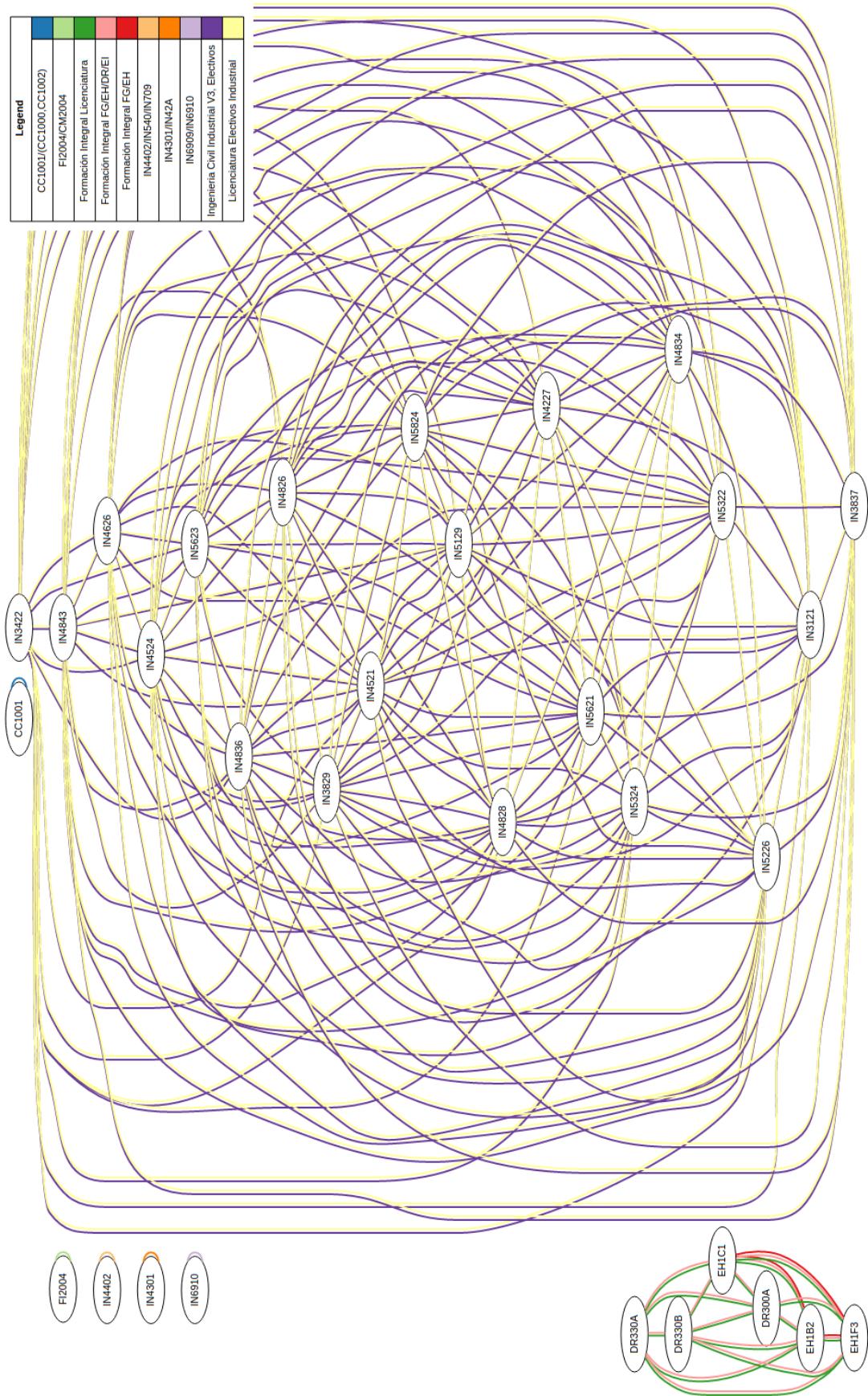


Figura 6.1: Grafo de un alumno de la carrera Ingeniería Civil Industrial.

Finalmente, si no se considera la carrera antes mencionada, se logró reducir en 10 veces la cantidad de elementos que son calculados en más de 15 segundos. Además, considerando la totalidad de elementos, se logró encontrar 64 falsos negativos siendo el 25 % de los cómputos reducidos desde 15 segundos más, hacia un valor menor. Además, se encontró que cerca del 4 % de los candidatos a mejorar sus recuentos presentaba una nota mayor a la calculada y por ende, era probable que podría titularse con una nota mayor o, incluso, con una distinción superior.

Conclusión

A partir de lo ya dicho en la sección de “Validación”, se logró formular un algoritmo capaz de mejorar la anterior automatización del recuento de UD’s reduciendo considerablemente los tiempos de gran parte de los planes, evidenciando de buena manera el comportamiento que sigue para próximas evaluaciones de los métodos e implementaciones realizadas.

Si bien se encontró un caso en que el comportamiento no es el óptimo, se logró esclarecer las razones constatando cómo una decisión administrativa que afecta la disposición de los ramos puede llegar a influir tanto a la descripción del avance curricular del alumno, como perjudicar el cálculo automatizado.

Como opinión, se considera que los cambios de la carrera de Ingeniería Civil Industrial poseen innovaciones en las metodologías o adecuación a las necesidades de los estudiantes, por lo tanto, son medidas atractivas que logran instaurarse dentro del normal desarrollo. Aun así, debe existir un equilibrio y consenso a nivel de universidad de la instauración de este tipo de medidas en el sentido de que actualmente, como se poseen dominios separados entre departamentos y Escuela, se debe velar por el normal funcionamiento local pero a su vez debe existir una cohesión para el trabajo en conjunto. Esto incluso implica el asesoramiento que deben tener ciertas implementaciones de normas y reglamentos, debido a la falta de visión en la posibilidad de implementación que pueden llegar a poseer. Se debe tener cierta certeza de la posibilidad de puesta en marcha antes de firmar un decreto.

Por otra parte, evaluando superficialmente el recuento de UD’s en versiones antiguas (por ejemplo en el plan “Ingeniería Civil Electricista v2”) se logró notar ciertos planes que no seguían un comportamiento lógico más que el llenar de alguna manera recuentos que no cumplían todas las reglas (como es el caso de los planes “Libres”). Esto también se debe a que antiguamente el actor principal del proceso tenía la potestad y mayor flexibilidad a la hora de decidir los alumnos con viabilidad de titularse. En cambio, éste, al ser un sistema automatizado posee mayor rigidez y, si bien en un inicio se poseen planes bien modelados, finalmente se incurre a “ensuciar” los planes de estudios a través del tiempo.

Un ejemplo de esto fue la inclusión, hace un tiempo atrás, de dos ramos de computación reemplazando el antiguo curso de programación, que si bien fue una excelente medida, el modelamiento inicial en el recuento de UD’s fue cuestionable y dadas conversaciones terminó perjudicando menormente el plan. Aun así, recuperando uno de los párrafos mencionados en el “Marco Teórico” acerca de la componente gigante que puede formarse en grafos aleatorios al ir agregando cada vez más elementos, si bien no es del todo el caso, el “ensuciar” de a poco la carrera puede provocar finalmente la generación de una gran componente gigante

implicando que parte de las mejoras provocadas en esta memoria, debido a la separación de componentes conexas se vea neutralizada.

Por lo anterior, sería útil la formación de herramientas que permitan guiar y asesorar los lineamientos de los planes de estudios para que, si bien permitan innovaciones dentro de él o la adición de elementos creativos, también se posea una mirada tecnológica con experiencia fundada de la evaluación de dichos planes y vayan analizando a través del tiempo su comportamiento.

Bibliografía

- [1] Universidad de Chile. Reglamento de estudios de la facultad de ciencias físicas y matemáticas (plan 2007). <http://escuela.ing.uchile.cl/normas-y-reglamentos/reglamento-de-estudios-plan-2007>. [En linea].
- [2] José Miguel Garrido. Diseño y construcción de un sistema de apoyo a la verificación del cumplimiento del plan de estudios en la facultad de ciencias físicas y matemáticas. Master's thesis, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, 2005.
- [3] Silvano Martello, Paolo Toth. *Knapsack Problems, Algorithms and Computer Implementations*. J. Wiley Sons, New York, 1990.
- [4] Silvano Martello, Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. In *Discrete Applied Mathematics*, pages 59–70. Elsevier Science Publishers B.V., North Holland, 1990.
- [5] Nelson Valdivia. Diseño de un sistema inteligente de recuento de unidades docentes para la escuela de ingeniería y ciencias. Master's thesis, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Santiago, 2003.

Apéndice A

Código de la Implementación

```
1  function _compConexas( $aprobados , $planes , $comp = array() ) {
2      $rep = array();
3      foreach( $planes as $p => $plan ) {
4          if( $plan[ PLA_TIPO ] == todo || ! $plan[ planes ] ) continue;
5          foreach( array_keys( $plan[ planes ] ) as $p2 ) $rep[$p2] = $p;
6      }
7
8      foreach( $aprobados as $codigo => $r ) {
9          if( $r[ pla_id ] || empty( $r[ planes ] ) ) || ( $comp && ! $comp[ ramos
10             ][ $codigo ] ) ) continue;
11         $min = min( array_keys( $r[ planes ] ) );
12         foreach( array_keys( $r[ planes ] ) as $p ) {
13             if( empty( $rep[$p] ) ) {
14                 $rep[$p] = $min;
15             } elseif( $rep[$p] != $min ) {
16                 $max = max( $rep[$p] , $min );
17                 $min = min( $rep[$p] , $min );
18                 foreach( array_keys( $rep ) as $k ) if( $rep[$k] == $max ) $rep[$k]
19                     = $min;
20             }
21         }
22         $com = array();
23         foreach( $aprobados as $codigo => $r ) {
24             if( $r[ pla_id ] || empty( $r[ planes ] ) ) || ( $comp && ! $comp[ ramos
25                 ][ $codigo ] ) ) continue;
26             $com[ $rep[ min( $r[ planes ] ) ] ][ ramos ][ $codigo ] = $codigo;
27             $com[ $rep[ min( $r[ planes ] ) ] ][ planes ] = array_merge( (array) $com[ $rep[ min( $r[ planes ] ) ] ][ planes ] , $r[ planes ] );
28         }
29
30         return $com;
31     }
32
33     function _marcarTODOs( $p , &$planes , $completar = FALSE ) {
34         $no_todo = $p[ PLA_TIPO ] != todo ;
35         $completar |= $no_todo;
```

```

36 $planes[ $p[ PLA_ID ] ][ _completar ] = $completar ? ( $no_todo ? $p[
37     cantidad ] : 0 ) : count( $p[ ramos ] );
38 foreach( $p[ planes ] as $pid ) {
39     if( ! isset( $planes[$pid][ CARR_TIPO_TITULO ] ) ) $planes[$pid][
40         CARR_TIPO_TITULO ] = $p[ CARR_TIPO_TITULO ]; //Propago
41         CARR_TIPO_TITULO
42     biaAPI::_marcarTODOs( $planes[$pid], $planes, $completar );
43 }
44
45 function procesar_recuento2( $rut, $carr_codigo, $c_plan ) {
46     global $_cont, $_cache;
47
48     $rut          = (int) $rut;
49     $carr_codigo = (int) $carr_codigo;
50     $c_plan       = (int) $c_plan;
51
52     if( ! $rut || ! $carr_codigo || ! $c_plan ) return array();
53
54     $pla_id = biaAPI::getPlanCarrera( $carr_codigo, $c_plan );
55     $planes = biaAPI::getPlanes( $pla_id );
56
57     if( ! $pla_id || empty( $planes ) ) {
58         KERNEL::mensaje( -No_hay_plan_asociado_a_esta_carrera );
59         return array();
60     }
61
62     $carrera = SIMBAD::getCarreras( $carr_codigo );
63     list( $aprobados, $actuales ) = biaAPI::getAprobados( $rut, $carrera[
64         CARR_TIPO_TITULO ] >= 3 ? 0 : $c_plan, $carr_codigo );
65
66     $carreras_alumno = UTIL::reindex( SIMBAD::getCarrerasAlumno( $rut ),
67         CARR_CODIGO );
68     if( ! empty( $carreras_alumno[$carr_codigo] ) ) {
69         $c = $carreras_alumno[$carr_codigo];
70         if( ! empty( $c[ fin ] ) && in_array( $c[ P_AL_E_PLAN_ALUM ], array( 5,
71             7, 8, 10, 11 ) ) ) {
72             foreach( $aprobados as $codigo => $ramo ) {
73                 if( $ramo[ anno ] > $c[ fin ][ anno ] ) {
74                     unset( $aprobados[$codigo] );
75                 }
76             }
77         }
78     }
79
80     // QUITO LA NOTA DE LOS RAMOS TERMINALES
81     if( $carrera[ CARR_TIPO_TITULO ] != 1 || ( SIMBAD::$db == facso && $
82         carr_codigo != 10286 ) ) {
83         $ramos_terminales = biaAPI::getRamosTerminales( $carr_codigo );
84         foreach( $aprobados as $codigo => $ramo ) {
85             if( $ramos_terminales[$codigo] && $ramo[ nota_final ] ) {
86                 $aprobados[$codigo][ nota_final ] = 0;
87                 $aprobados[$codigo][ nota_texto ] .= <em>*</em> ;
88             }
89         }
90     }

```

```

85
86 // MARCAR RAMOS CON LISTAS
87 foreach( $planes as $pid => $plan ) {
88     foreach( $aprobados as $codigo => $ramo ) {
89         if( $plan[ ramos ][$codigo] ) {
90             $aprobados[$codigo][ planes ][$pid] = $pid;
91             continue;
92         }
93         foreach( $plan[ ramos ] as $r ) {
94             if( ! $r[ eq ] ) continue;
95             $codigos = explode( / , $r[ eq ] );
96             if( in_array( $codigo , $codigos ) ) {
97                 $aprobados[$codigo][ planes ][$pid] = $pid;
98                 $aprobados[$codigo][ eq ][$pid] = $r[ codigo ];
99             }
100        }
101    }
102    foreach( $actuales as $codigo => $ramo ) {
103        if( $plan[ ramos ][$codigo] ) {
104            $actuales[$codigo][ planes ][$pid] = $pid;
105            continue;
106        }
107        foreach( $plan[ ramos ] as $r ) {
108            if( ! $r[ eq ] ) continue;
109            $codigos = explode( / , $r[ eq ] );
110            if( in_array( $codigo , $codigos ) ) {
111                $actuales[$codigo][ planes ][$pid] = $pid;
112                $actuales[$codigo][ eq ][$pid] = $r[ codigo ];
113            }
114        }
115    }
116 }
117
118 // MARCAR LOS TODOS "LIMPIOS" Y PROPAGAR CARR_TIPO_TITULO
119 biaAPI::_marcarTodos( $planes[$pla_id] , $planes );
120
121 // ORDENAR APROBADOS POR NOTA FINAL + UD
122 UTIL::sort( $aprobados , _nota_final , _ud );
123
124 // ASIGNAR RAMOS EN PLANES TODO "LIMPIOS"
125 $cumple_acum = TRUE;
126 foreach( $planes as $pid => $p ) {
127     if( $p[ PLA_TIPO ] != todo || ! $p[ _completar ] ) continue;
128     foreach( $aprobados as $codigo => $r ) {
129         if( $r[ pla_id ] || ! $r[ planes ][$pid] ) continue;
130
131         biaAPI::_asignarRamo( $codigo , $pid , $aprobados , $planes );
132         if( ! $planes[$pid][ _completar ] ) break;
133     }
134     if( $planes[$pid][ _completar ] ) $cumple_acum = FALSE;
135 }
136
137 foreach( $aprobados as $c => $r ) {
138     if( $r[ pla_id ] || count( $r[ planes ] ) > 1 ) continue;
139     else if( ! $r[ planes ] ) {
140         $no_usados[$c] = $r;

```

```

141         unset( $aprobados[$c] );
142     }
143 }
144
145 // PROCESAMIENTO DEL ALGORITMO
146 $_tiempo = microtime( TRUE );
147 $_cache = array();
148 list( $aprobados, $planes ) = biaAPI:: _recuento( $aprobados, $planes, $pla_id );
149 $planes2 = $planes;
150 biaAPI:: _postorder( $pla_id, $planes, TRUE );
151
152             //POST-PROCESAMIENTO
153 foreach( $aprobados as $c => $r ) {
154     if( ! $r[ pla_id ] || $r[ pla_id ] == NO_USAR ) {
155         $mejor_n_carrera = $r[ nota_final ] >= $planes[$pla_id][ promedio_carrera ];
156         $mejor_n_general = $r[ nota_final ] >= $planes[$pla_id][ promedio ];
157         if( ( $mejor_n_carrera || $mejor_n_general ) && $r[ planes ] ) {
158             $candidato = ;
159             foreach( $r[ planes ] as $pid ) {
160                 if( ( $planes[$pla_id][ CARR_TIPO_TITULO ] == 2 && $planes[$pid][ CARR_TIPO_TITULO ] == 2 && $mejor_n_carrera ) || ( $planes[$pla_id][ CARR_TIPO_TITULO ] != 2 && $mejor_n_general ) ) {
161                     $candidato = $pid;
162                     break;
163                 }
164             }
165             if( $candidato ) {
166                 biaAPI:: _asignarRamo( $c, $candidato, $aprobados, $planes2 );
167                 $planes = $planes2;
168                 biaAPI:: _postorder( $pla_id, $planes, TRUE );
169                 continue;
170             }
171         }
172         unset( $aprobados[$c][ pla_id ], $r[ pla_id ] );
173         $no_usados[ $r[ codigo ] ] = $r;
174     }
175 }
176
177 $_tiempo = number_format( max( microtime( TRUE ) - $_tiempo, 0.001 ), 3,
178 , , , );
179 foreach( $_cache as $k => $v ) unset( $GLOBALS[ _cache ][ $k ][ 0 ], $GLOBALS[ _cache ][ $k ][ 1 ], $GLOBALS[ _cache ][ $k ][ 2 ] );
180 return array( $pla_id, $planes, $aprobados, $actuales, $no_usados, $_cont,
181 $_tiempo );
182
183 function _recuento( $aprobados, $planes, $pla_id, $comp = array(), $ultimo =
184 " " ) {
185     $comps = biaAPI:: _compConexas( $aprobados, $planes, $comp );
186     foreach( $comps as $c ) list( $aprobados, $planes ) = biaAPI:: _resolver( $c, $aprobados, $planes, $pla_id );
187     return array( $aprobados, $planes );

```

```

188     }
189
190     function _resolver( $comp, $aprobados, $planes, $pla_id ) {
191         $ramo = array_shift( $comp[ ramos ] );
192         if( ! $ramo ) return array( $aprobados, $planes );
193         $ramo = $aprobados[$ramo];
194
195         $soluciones = array();
196         foreach( $ramo[ planes ] as $p ) {
197             if( $planes[$p][ _completar ] <= 0 ) continue;
198             $old = biaAPI:: _asignarRamo( $ramo[ codigo ], $p, $aprobados, $planes );
199             if( ! $old ) continue;
200             $soluciones[] = biaAPI:: _recuento( $aprobados, $planes, $pla_id, $comp,
201                 $ramo[ codigo ] );
202             list( $aprobados, $planes ) = $old;
203         }
204         if( $soluciones ) return biaAPI:: _mejorSolucion( $soluciones, $pla_id );
205
206         $aprobados[$ramo[ codigo ]][ pla_id ] = NO_USAR ;
207         return biaAPI:: _recuento( $aprobados, $planes, $pla_id, $comp, $ramo[
208             codigo ] );
209     }
210
211     function _mejorSolucion( $soluciones, $pla_id ) {
212         global $_cont;
213
214         ++$_cont;
215         list( $aux_aprob, $aux_planes ) = array_shift( $soluciones );
216         if( ! $soluciones ) return array( $aux_aprob, $aux_planes );
217
218         $aux_calc = $aux_planes; $aux2_calc = array();
219         biaAPI:: _postorder( $pla_id, $aux_calc );
220         while( list( $aux2_aprob, $aux2_planes ) = array_shift( $soluciones ) ) {
221             $aux2_calc = $aux2_planes;
222             biaAPI:: _postorder( $pla_id, $aux2_calc );
223
224             /*MEJOR ENTRE DOS*/
225             ++$_cont;
226             $_cumple = $aux_calc[$pla_id][ ok ];
227             $actual_cumple = $aux2_calc[$pla_id][ ok ];
228             if( $_cumple && ! $actual_cumple ) continue;
229
230             $_carrera = $aux2_calc[$pla_id][ CARR_TIPO_TITULO ] == 2;
231             $mejor_cumple = ! $_cumple && $actual_cumple;
232             $mejor_oks = ! $actual_cumple && ( $aux2_calc[$pla_id][ oks ] > $aux_calc[$pla_id][ oks ] );
233             $igual_oks = ! $actual_cumple && $aux2_calc[$pla_id][ oks ] == $aux_calc[$pla_id][ oks ];
234             $igual_prom_c = $igual_oks && $aux2_calc[$pla_id][ promedio_carrera ] ==
235                 $aux_calc[$pla_id][ promedio_carrera ];
236             $mejor_prom_c = $igual_oks && $aux2_calc[$pla_id][ promedio_carrera ] >
237                 $aux_calc[$pla_id][ promedio_carrera ];
238             $mejor_prom = $igual_prom_c && $aux2_calc[$pla_id][ promedio ] > $aux_calc[$pla_id][ promedio ];
239             $mejor_nota = $_cumple && $actual_cumple && ( ( $_carrera && $aux2_calc[
```

```

        $pla_id ][ promedio_carrera ] >= $aux_calc[$pla_id][ promedio_carrera
        ] ) || ( ! $carrera && $aux2_calc[$pla_id][ promedio ] >= $aux_calc[
        $pla_id ][ promedio ] ) );
237    if( $mejor_cumple || $mejor_oks || $mejor_prom_c || $mejor_prom || $mejor_nota ) {
238        $aux_aprob = $aux2_aprob;
239        $aux_planes = $aux2_planes;
240        $aux_calc = $aux2_calc;
241    }
242}
243
244    return array( $aux_aprob, $aux_planes );
245}
246
247 function _asignarRamo( $codigo, $pid, &$aprobados, &$planes ) {
248    $old = array( $aprobados, $planes );
249    if( $eq = $aprobados[$codigo][ eq ][$pid] ) {
250        if( ( $aprobados[$eq] && $aprobados[$eq][ pla_id ] ) || ! $planes[$pid][ ramos ][ $eq ] ) return FALSE;
251        $aprobados[$codigo][ nombre ] .= <em>(Equivalente_a_ . $eq. )</em>;
252        $aprobados[$codigo][ ud ] = max( $aprobados[$codigo][ ud ], getUD( $eq ) );
253        unset( $planes[$pid][ ramos ][ $eq ] );
254        if( $aprobados[$eq] ) $aprobados[$eq][ planes ] = array();
255    }
256    $aprobados[$codigo][ pla_id ] = $pid;
257    $planes[$pid][ _completar ] == ( $planes[$pid][ PLA_TIPO ] == ud ? $aprobados[$codigo][ ud ] : 1 );
258    $planes[$pid][ usados ][$codigo] = $codigo;
259    $planes[$pid][ ud ] += $aprobados[$codigo][ ud ];
260    $planes[$pid][ cont ] += 1;
261    $planes[$pid][ semestre ] = max( (int) $planes[$pid][ semestre ], $aprobados[$codigo][ semestre ] );
262    if( $aprobados[$codigo][ nota_final ] ) {
263        $planes[$pid][ notas ][$codigo] = floatval( $aprobados[$codigo][ nota_final ] );
264        $planes[$pid][ notas_ponderadas ][$codigo] = floatval( $aprobados[$codigo][ nota_final ] ) * ( (int) $aprobados[$codigo][ ud ] );
265        $planes[$pid][ ud_ponderada ] = (int) $planes[$pid][ ud_ponderada ] + $aprobados[$codigo][ ud ];
266        if( $planes[$pid][ _completar ] <= 0 ) $planes[$pid][ prom_local ] = promedio( $planes[$pid][ notas ] );
267    }
268    if( $planes[$pid][ CARR_TIPO_TITULO ] == 2 ) {
269        $planes[$pid][ notas_carrera ] = $planes[$pid][ notas ];
270        $planes[$pid][ ud_carrera ] = $planes[$pid][ ud ];
271    }
272
273    return $old;
274}
275
276 function _postorder( $p, &$planes, $final = FALSE ) {
277    if( $planes[$p][ planes ] ) {
278        foreach( $planes[$p][ planes ] as $pid ) biaAPI::_postorder( $pid, $planes );
279    }

```

```

280     foreach( $planes[$p][ planes ] as $pid ) {
281         $planes[$p][ notas ] = (array) $planes[$p][ notas ] + (array) $planes[
282             $pid ][ notas ];
283         $planes[$p][ notas_ponderadas ] = (array) $planes[$p][
284             notas_ponderadas ] + (array) $planes[$pid][ notas_ponderadas ];
285         $planes[$p][ semestre ] = max( (int) $planes[$p][ semestre ], (int) $[
286             planes[$pid][ semestre ] );
287         $planes[$p][ diff ] += abs( $planes[$pid][ diff ] );
288         $planes[$p][ oks ] = (int) $planes[$p][ oks ] +(int) $planes[$pid][
289             oks ];
290         if( $planes[$pid][ CARR_TIPO_TITULO ] >= 2 ) {
291             $planes[$p][ notas_carrera ] = (array) $planes[$p][ notas_carrera ] [
292                 + (array) $planes[$pid][ notas_carrera ];
293             $planes[$p][ ud_carrera ] = (int) $planes[$p][ ud_carrera ] + (int) [
294                 $planes[$pid][ ud_carrera ];
295         }
296     }
297 }
298
299 $planes[$p][ ud_aporte ] = $planes[$p][ maximo ] ? min( $planes[$p][
300     maximo ], $planes[$p][ ud ] ) : $planes[$p][ ud ];
301 $planes[$p][ promedio ] = promedio( (array) $planes[$p][ notas ], $final ?
302     1 : 10 );
303 $planes[$p][ promedio_ponderado ] = $planes[$p][ ud ] ? number_format(
304     array_sum( (array) $planes[$p][ notas_ponderadas ] ) / $planes[$p][ ud ],
305     1, ., . ) : 0.0;
306 $planes[$p][ promedio_carrera ] = promedio( (array) $planes[$p][
307     notas_carrera ], $final ? 1 : 10 );
308 $planes[$p][ diff ] += $planes[$p][ cantidad ] - ( $planes[$p][ tipo ] ==
309     ud ? $planes[$p][ ud ] : $planes[$p][ cont ] );
310 $planes[$p][ usados ] = (array) $planes[$p][ usados ];
311
312 if( $planes[$p][ ok ] && $planes[$p][ planes ] ) {
313     foreach( $planes[$p][ planes ] as $k ) {
314         if( ! $planes[$k][ ok ] ) ++$planes[$p][ oks ];
315         $planes[$k][ ok ] = TRUE;
316     }
317 }
318 }
```