Universidad Autónoma Metropolitana

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Tecnológico

Búsqueda armónica para resolver un problema de asignación de unidades de enseñanza y aprendizaje

Garduño Villaseñor Tanaidy
208301575

Asesores:
Dr. Eric Alfredo Rincón García
Profesor Asociado
Departamento de Sistemas

Dr. Marco Antonio Heredia Velasco
Profesor Titular
Departamento de Sistemas

Trimestre 2014 Primavera
25 de Agosto 2014 – version 4.0

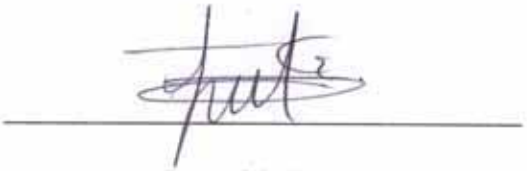
Yo, Eric Alfredo Rincón García , declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma del asesor

Yo, Marco Antonio Heredia Velasco , declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma del asesor

Yo, Tanaidy Garduño Villaseñor, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Firma del alumno

RESUMEN

El plan de estudios de la carrera de Ingeniería en Computación, ha sido elaborado de manera progresiva con el objetivo de que el alumno avance en la acumulación de conocimientos conforme va inscribiendo Unidades de Enseñanza y Aprendizaje (UEA). Sin embargo, la flexibilidad del mismo, permite que el alumno inscriba materias de distintos trimestres sin reflexionar en la importancia del conocimiento previo que debe tener para su siguiente trimestre. Esto puede ocasionar que el alumno deje inconclusa la materia, incrementando su índice de no aprobación, y en consecuencia, repercutir en la eficiencia terminal. Así mismo, existen materias que el alumno no siempre considera y que pueden ser esenciales para inscribir los cursos del siguiente bloque.

En este proyecto de Integración se adaptó la técnica heurística "Búsqueda Armónica“ (Harmony Search), para generar propuestas, que indiquen el trimestre en el cual debe cursarse cada una de las UEA seleccionada por un estudiante, de la carrera de Ingeniería en Computación de la Universidad Autónoma Metropolitana unidad Azcapotzalco. Para cada propuesta realizada, el algoritmo busca minimizar el número de trimestres requeridos, tomando en consideración restricciones impuestas por la universidad.

La finalidad de este proyecto es ofrecer una herramienta computacional, que le indique al alumno el trimestre en el cual debe cursar las UEA que ha seleccionado, y terminar en un tiempo adecuado, para lo cual se creó un banco de instancias académicas a partir del plan de estudios antes mencionado, que simulan la selección de UEA que podrían realizar un conjunto de estudiantes.

El uso de la técnica heurística Búsqueda Armónica en estas instancias fue útil para determinar su eficiencia en este tipo de problemas en comparación con otros métodos del estado del arte. Además, esta técnica permitió obtener diversas soluciones de buena calidad que ayudaron a determinar qué tan rígido es el plan de estudios.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE LISTADOS DE CÓDIGO

1 INTRODUCCIÓN

El problema de currículo académico consiste en asignar Unidades de Enseñanza y Aprendizaje (UEA) en un número preestablecido de ciclos escolares de tal forma que se promueva una carga académica equilibrada para cada trimestre, respetando ciertas restricciones como Seriación, Créditos permitidos por trimestre, y Créditos Mínimos para tomar una UEA. Encontrar soluciones válidas para este tipo de problemas puede requerir de mucho tiempo, por lo cual resulta útil contar con herramientas automatizadas eficientes capaces de generar soluciones de buena calidad. Por lo anterior, se plantea el uso de una técnica heurística para diseñar un algoritmo, capaz de generar soluciones que satisfacen las restricciones antes mencionadas, y que de forma adicional requieran el mínimo número de trimestres para realizar la asignación de las UEA elegidas por el estudiante.

El producto final de este proyecto es un programa desarrollado en el lenguaje C++ que propone un algoritmo basado en la metaheurística búsqueda armónica, que es una estrategia de propósito general que consiste en procedimientos iterativos que combinan de forma inteligente distintas soluciones para explorar y explotar adecuadamente el espacio de búsqueda.

Los resultados obtenidos muestran la capacidad de esta técnica heurística para resolver problemas de currículo académico.

2

ANTECEDENTES

Los planes de estudio de la Universidad Autónoma Metropolitana unidad Azcapotzalco (UAM-A) están formados por UEA que deben ser cursadas por los estudiantes en un número preestablecido de trimestres. Cada UEA tiene asociado un número de créditos y en algunos casos requisitos para poder cursarla:

1) Seriación. La inscripción a algunas UEA está sujeta a la acreditación previa de otras unidades.

2) Mínimo número de créditos. La inscripción a algunas UEA está sujeta a que el estudiante haya obtenido cierto número de créditos por las UEA acreditadas.

Del mismo modo, se establece un número máximo de 52 créditos que puede llevar un estudiante durante cada trimestre.

Recientemente se realizaron modificaciones en algunos planes de estudio de la UAM-A, dando como resultado la desaparición y creación de UEA, lo cual afectó la seriación y el orden en que deben ser cursadas. Estos cambios hicieron necesaria una revisión para establecer una nueva distribución de las UEA y para determinar si los nuevos planes podrían ser completados en el tiempo establecido por la UAM-A. Este trabajo fue realizado manualmente para cada plan de estudio, y requirió un tiempo el tratar de diseñar una solución capaz de satisfacer las condiciones establecidas **1)-2)**.

Gran parte de este esfuerzo puede evitarse mediante el uso de un algoritmo de asignación de tareas, el cual ayude a determinar una solución factible, es decir una solución que permita al estudiante cursar todas las UEA requeridas en el tiempo establecido. Sin embargo, la complejidad de este tipo de problemas hace recomendable el uso de técnicas heurísticas para obtener soluciones de buena calidad en tiempos de cómputo aceptables.

De hecho, durante la última década se ha presentado un crecimiento sostenido en el campo de los algoritmos meta-heurísticos para la búsqueda y optimización de problemas en diferentes áreas, tanto de investigación como de aplicaciones. Tales métodos, debido a su capacidad de encontrar soluciones muy cercanas a las óptimas dentro de un periodo de tiempo razonable, han empezado a ser usados para resolver diferentes problemas de ingeniería, los cuales tradicionalmente usaban otros enfoques. En este proyecto se propone emplear la técnica heurística Búsqueda Armónica (BA), la cual usa como metáfora el proceso de improvisación musical que ocurre cuando un músico busca la armonía óptima. De esta manera en BA una posible solución es similar a una armonía, mientras que sus operadores simulan

el proceso de improvisación. En comparación con otros algoritmos meta-heurísticos, BA presenta varias ventajas como son la utilización de pocos parámetros de configuración y su rapidez. Considerando estas ventajas el algoritmo de BA ha sido satisfactoriamente aplicado para resolver una gran cantidad de problemas complejos de optimización y se considera que puede ser una herramienta adecuada para el problema analizado en este trabajo.

3

JUSTIFICACIÓN

Con frecuencia, para determinar el orden en el que deben tomarse los cursos en una licenciatura de manera factible y al mismo tiempo minimizando el número de trimestres, se recurre a procesos manuales que requieren gran cantidad de tiempo. Por lo tanto, diseñar una herramienta capaz de generar soluciones de forma automática y eficiente se ha convertido en un objetivo valioso. Sin embargo, se ha demostrado que este problema es NP-Duro, por lo que el uso de métodos exactos puede ser inadecuado en instancias de la vida real, ya que su ejecución puede requerir tiempos de cómputo demasiado largos.

Por lo anterior, el uso de técnicas heurísticas es una opción que permite encontrar soluciones de buena calidad en tiempos de cómputo aceptables. En este proyecto se diseñará una herramienta capaz de generar, de manera automática, soluciones que respetan las condiciones establecidas por la UAM-A, y que distribuyen las UEA en el mínimo número de trimestres necesarios para completar la licenciatura. El algoritmo diseñado se basa en Búsqueda Armónica, una de las técnicas heurísticas existentes en la actualidad, que ha destacado debido a sus excelentes resultados. Las instancias empleadas en este proyecto se generarán a partir de los planes de estudios de las carreras de Ingeniería en Computación e Ingeniería Física de la UAM-A. Esto implica que las soluciones obtenidas para instancias que contengan el mínimo número de UEA requerido para que un estudiante complete su licenciatura, podrán coincidir con los diagramas de seriación de la carrera, en el caso de que 13 trimestres sean el mínimo tiempo requerido para cursarlas. Asimismo, utilizaremos la existencia de un proyecto de integración ya concluido, para comparar el desempeño o la calidad de las soluciones obtenidas en esta propuesta.

Se debe destacar que hasta donde se tiene conocimiento Búsqueda Armónica nunca se ha empleado para resolver el problema antes mencionado.

4 OBJETIVOS

4.1 OBJETIVO GENERAL

Adaptar e implementar un algoritmo inspirado en la heurística Búsqueda Armónica para resolver algunas instancias del problema de asignación de unidades de enseñanza y aprendizaje.

4.2 OBJETIVOS ESPECÍFICOS

1. Adaptar la técnica heurística Búsqueda Armónica al problema de asignación de Unidades de Enseñanza y Aprendizaje.
2. Implementar un algoritmo basado en la adaptación de Búsqueda Armónica.
3. Calibrar el algoritmo sobre un conjunto reducido de instancias.
4. Analizar los resultados y comparar el desempeño de la técnica propuesta con el de algoritmos actualmente operantes.

5

MARCO TEÓRICO

El problema de asignación de horarios es la asignación, sujeta a restricciones, de recursos a ciertas tareas dentro de periodos de tiempo preestablecidos, de tal forma que se satisfacen un conjunto de objetivos. Este tipo de problemas aparece en diferentes actividades cotidianas, como generar turnos de enfermeras [1], horarios en medios de transporte [2], en asignación de equipos de trabajo [3]. También se puede encontrar en instituciones educativas de nivel superior, donde implica la asignación de profesores, aulas, laboratorios, cursos, exámenes, entre otros, a horarios establecidos [4].

Un caso particular de este tipo de problemas es conocido como currículo académico, en el cual se busca determinar el periodo escolar en el que un estudiante deberá cursar cada una de las materias requeridas para poder completar su licenciatura, de tal forma que se cumplan con restricciones, como seriaciones, y se optimicen objetivos, como cargas académicas balanceadas [5]. Esta clase de problemas puede verse como un caso particular del problema de empaquetamiento con restricciones de precedencia, del cual hereda la complejidad computacional, que ha demostrado ser NP-Duro [6], por lo que el uso de métodos exactos puede ser inadecuado en instancias de la vida real. Por lo tanto, las técnicas heurísticas se han convertido en la herramienta más adecuada para resolver este tipo de problemas. De hecho, en los trabajos mencionados anteriormente se emplearon técnicas como búsqueda local iterada, un algoritmo mimético basado en búsqueda Tabú y apareamiento de abejas melíferas.

En este proyecto se propone emplear un algoritmo basado en la técnica heurística Búsqueda Armónica para resolver un problema de currículo académico, empleando instancias generadas a través de los planes de estudio de las carreras de Ingeniería en Computación e Ingeniería Física de la Universidad Autónoma Metropolitana, unidad Azcapotzalco.

6 DESARROYO DEL PROYECTO

6.1 DESCRIPCIÓN DE LA TÉCNICA BÚSQUEDA ARMONICA

La forma básica del algoritmo Búsqueda Armónica consiste en tres etapas: inicialización, improvisación de nuevas armonías (generación de nuevas soluciones) y la actualización de la memoria (HM).

Durante todo el proceso de optimización, el algoritmo trabaja con un grupo de soluciones almacenadas en un conjunto llamado Memoria Armónica (Harmony Memory "HM"). Cada solución es considerada una "armonía", y es representada por un vector n-dimensional, donde n es el número de UEA en la instancia analizada. Las soluciones iniciales son generadas al asignar un trimestre aleatorio, entre 1 y 36, a cada UEA, y son almacenadas dentro de la memoria HM. Cada una de estas soluciones es evaluada por la función objetivo y su costo es guardado en memoria para futuras comparaciones. El número de soluciones en HM es determinado por el usuario al inicio de cada ejecución.

Para generar una solución nueva, NS, se recurre a las soluciones almacenadas en HM. Para cada UEA, i, se elige de forma aleatoria una solución en HM, SA_i, de esta forma en la solución NS la UEA i es asignada al mismo trimestre que en la solución SA_i.

Posteriormente se realiza un proceso que ayuda a diversificar la búsqueda del algoritmo, para lo cual se pueden realizar una de las siguientes técnicas. (1) "Ajuste de tono", lo cual implica sumar o restar un valor aleatorio al trimestre de algunas UEA. (2) "Nuevos tonos", que consiste en generar valores aleatorios para la asignación de algunas UEA.

Cuando la nueva solución ha sido construida se evalúa mediante la función objetivo. Si el costo de la nueva solución es mejor que el peor costo de las soluciones contenidas en HM, la peor solución dentro de la memoria es reemplazada, de lo contrario no existirá cambio alguno, esto se realiza mediante le Consideración de Cambio de la Memoria Armónica en (Harmony Memory Considering Rate, "HMCR").

Este proceso se realiza hasta cumplir el criterio de paro.

En la Figura 1 se observa la relación que existe entre los diferentes módulos explicados anteriormente.

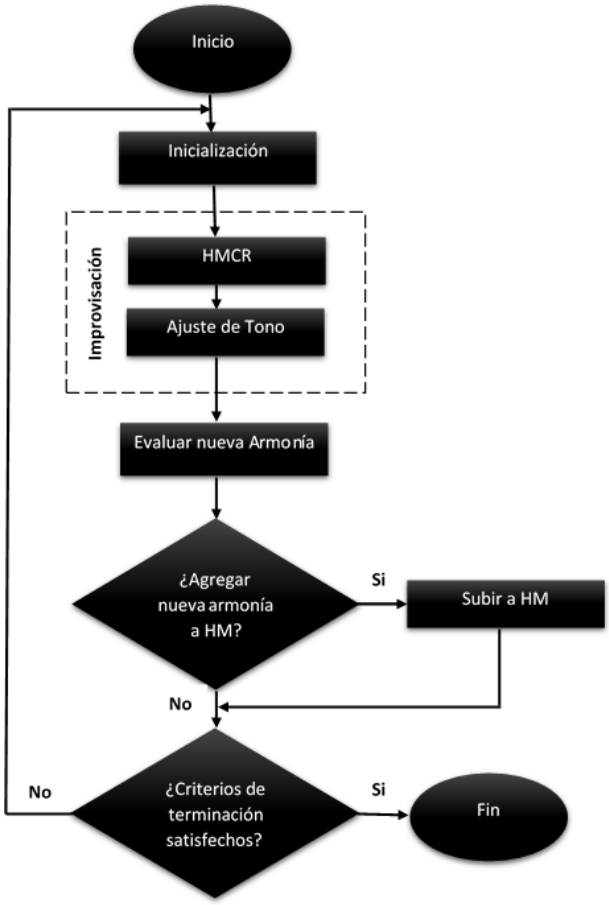


Figura 1: Diagrama de flujo Búsqueda Armónica.

6.2 DESCRIPCIÓN DEL PROBLEMA

De acuerdo al plan de estudios vigente en la Universidad Autónoma Metropolitana, se requiere que el alumno cumpla con un mínimo de créditos para poder concluir su carrera. Los alumnos acumulan créditos inscribiendo y cursando diversas Unidades de Enseñanza y Aprendizaje las cuales, de ser aprobadas, aportarán una cantidad determinada de créditos al alumno. La carrera se divide en unidades lectivas llamadas trimestres los cuales tienen un límite máximo de créditos, por lo que todas las UEA del plan de estudios deberán ser distribuidas a lo largo de estos trimestres.

Las UEA disponibles se pueden clasificar por el tipo de restricciones que deben cumplirse para poder ser inscritas/cursadas.

R1 UEA con seriación: requiere haber cursado determinadas UEA para poder ser inscrita.

R2 UEA con requisitos de créditos mínimos: exige el haber acumulado una cierta cantidad de créditos mínimos para poder ser inscrita.

R3 UEA sin requisitos: UEA que no requiere haber cursado alguna UEA ni haber acumulado una cierta cantidad de créditos.

Además de las anteriores, se agregó una restricción adicional:

R4 Límite superior. Se calculó el trimestre máximo en el cual una UEA puede tomarse. El número máximo de trimestres empleados por el algoritmo es 36, debido a que la UAM tiene como límite, un período de 12 años para terminar la licenciatura en la modalidad de medio tiempo.

6.3 DETALLES DEL PROGRAMA

El algoritmo se implementó utilizando el lenguaje de programación C++ y fue ejecutado en el IDE Dev C++ versión 4.9.9.2. El programa lee los datos de tres archivos de texto:

1. Archivo Datos: contiene las claves de todas las UEA con sus respectivos créditos.
2. Archivo Seriación: contiene las claves de las UEA y las claves de estas UEA requeridas para poder cursarlas.
3. Archivo Créditos Mínimos: contiene las claves y créditos de las UEA que necesitan cierto número de créditos mínimos para poder ser inscrita.

De estos archivos de texto se crearon en memoria tres arreglos asociativos donde cada una de sus entradas contiene una representación de cada UEA. Esta representación almacena los atributos asociados a cada UEA como, clave, créditos asociados, UEA requeridas y créditos requeridos.

El proyecto se divide en 7 archivos los cuales se dividieron en archivos con extensión .h, estos contendrán las funciones asociadas a ciertas clases creadas y archivos con la extensión .cpp, en estos archivos se encuentra la implementación de las funciones. Esto se realizó con la finalidad de tener una mejor organización y limpieza del programa.

6.3.1 *UEA.h y UEA.cpp*

Se creó una clase llamada UEA con funciones que permiten recibir y acceder a los elementos de este tipo como lo son las claves y créditos de una UEA.

6.3.2 *Seriacion.h y Seriacion.cpp*

Se creó una clase Seriacion, dentro de esta clase se crea una lista llamada claves de tipo entero, la cual contendrá todas las claves de las UEA requeridas para poder inscribir alguna UEA.

6.3.3 *Solucion.h y Solucion.cpp*

Dentro de estos archivos se tienen diferentes funciones las cuales son necesarias para poder generar una solución. Se crea una clase tipo solución, las funciones más importantes son las relacionadas a la función objetivo.

6.3.4 *Programa_Principal.cpp*

En este archivo se leen los tres archivos de texto y se crean listas para cada uno de ellos, para almacenar sus datos dentro de estas listas. Se implementa el código necesario para generar las soluciones que el usuario desea y regresar en pantalla la mejor solución que nos proporcione está técnica.

El programa se detendrá después de haber cumplido las 80000 iteraciones.

6.4 FUNCIÓN OBJETIVO

Para calcular el costo de cada solución, se evalúa el vector asociado a esta. La función objetivo es el resultado de la suma de 4 costos listados a continuación.

1. Trimestre Máximo: tiene como objetivo encontrar el mayor trimestre dentro de una solución y sumar ese valor a la función objetivo.
2. Violaciones por Créditos Excedidos: para cada trimestre de una solución se verifica el número de créditos acumulados, si excede los 52 créditos permitidos por la UAM, se suma a la función objetivo la diferencia entre estos valores.
3. Violaciones por Créditos Mínimos: si una UEA requiere cierta cantidad de créditos, se calcula la suma de los créditos acumulados hasta un trimestre anterior al cual fue asignado esta UEA. Si los créditos acumulados son menores a los créditos requeridos, se realiza la diferencia entre estas dos cantidades y se suma a la función objetivo.
4. Violaciones por Seriación: se revisa que las UEA con requisitos de seriación se ubiquen en trimestres posteriores con respecto a las UEA con las que están seriadas, si no es así, se suman las diferencias entre los trimestres de las UEA que no cumplan con la seriación.

6.5 PRUEBAS

Las pruebas aquí descritas se realizaron en una computadora con las siguientes características:

1. Procesador: Intel Core i3
2. Memoria RAM: 4 GB DDR3
3. Sistema Operativo: Windows 7

6.5.1 *Calibración del algoritmo*

Para calibrar el algoritmo se empleó el método de fuerza bruta. Este procedimiento consiste en establecer un periodo t, donde, de manera inicial se tiene una configuración arbitraria de los parámetros del algoritmo, denotada como θ, la cual es utilizada para ejecutar el algoritmo, evaluando los resultados obtenidos; posteriormente se altera uno de los parámetros de la configuración θ dando como resultado una configuración θ_1, dicha configuración es ocupada para la ejecución del algoritmo y se evalúan los resultados. Una vez evaluadas ambas configuraciones, se selecciona aquella configuración que genere los mejores resultados. Varios autores explican que el procedimiento de fuerza bruta es un proceso iterado de prueba y error.

6.5.2 *Pruebas realizadas*

Se seleccionaron las instancias del plan de estudios de Ingeniería Física y del plan de estudios de Ingeniería en Computación, para las cuales se ejecutaron 20 corridas utilizando el método Búsqueda Armónica descrito anteriormente, y realizando 80000 llamadas a la función objetivo y determinando para cada corrida, el tiempo de ejecución y la calidad de la solución entregada.

En la sección de Anexos, podremos encontrar algunas tablas que contienen las soluciones obtenidas, para las diferentes instancias utilizadas, donde:

1. Trimestres Empleados: es el número de trimestres en el que puedes terminar la licenciatura.

2. Factible: nos indica si la solución es aplicable, para lograr terminar la licenciatura en un tiempo adecuado(Se considera factible si el número máximo de trimestres empleados es 13)

3. MaxExcedidos: corresponde a una violación, el número obtenido nos dice en cuantos trimestres nos hemos pasado de los créditos permitidos.

4. MaxMinimos: corresponde a una violación, el número obtenido hace referencia a cuantas UEA requieren de tener un mínimo de créditos para poder cursarlas.

5. MaxSeriacion: corresponde a una violación, el número obtenido representa las UEA que no cumplieron con la seriación.
6. Tiempo de Ejecución: es el tiempo en que tarda en generar una solución.

7 ANÁLISIS Y DISCUSIÓN DE RESULTADOS

Para determinar la eficiencia del algoritmo propuesto en este proyecto, se consideraron 7 instancias creadas a partir del los planes de estudio de las carreras de Ingeniería en Computación, y de Ingeniería Física de la Universidad Autónoma Metropolitana Unidad Azcapotzalco. Cada una de estas instancias analizadas se presenta en los anexos A y B.

Para cada instancia se realizaron 20 ejecuciones del algoritmo, los resultados de cada corrida se encuentran en los anexos C, D, E y F. Para cada corrida se indica el número máximo de trimestres empleado por la mejor solución reportada por el algoritmo, así como las violaciones cometidas, y el tiempo de ejecución requerido.

Se observó que cada una de las instancias presenta una dificultad diferente, lo cual se aprecia tanto en el mínimo número de trimestres requerido por una solución factible, como por la cantidad de soluciones sin violaciones que pudo generar el algoritmo. En las tablas 1, 2 y 3 se presenta un resumen de los resultados obtenidos. Se aprecia que en todos los casos se requirieron 12 o menos trimestres, y para todas las instancias se emplearon menos de 25 segundos en promedio.

Se observa que las instancias más complicadas fueron las de "Algoritmos e Inteligencia Artificial" e "Instrumentación II" para las cuales sólo en una corrida se obtuvo una solución válida con el mínimo número de trimestres, mientras que la más sencilla fue instrumentación, para la cual se obtuvieron, en 10 corridas, soluciones válidas que requerían de 11 trimestres.

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Algoritmos e Inteligencia Artificial	12	1	20.4235 seg
Mecatrónica	12	9	19.8969 seg
Sistemas de Información	12	3	20.8139 seg

Tabla 1: Ingeniería en Computación.

Para determinar la eficacia del algoritmo propuesto se compararon los resultados obtenidos con los reportados en el proyecto de integración [7]. Es importante destacar que en dicho proyecto se empleó un algoritmo basado en la técnica heurística Optimización por Enjambre de Partículas, y que las instancias empleadas, Instrumentación y Tecnología I y II, son las mismas que se reportan en este trabajo. El

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Instrumentación I	11	10	18.98935 seg
Instrumentación II	11	1	21.00395 seg

Tabla 2: Instrumentación I y II.

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Tecnología I	12	5	21.0254 seg
Tecnología II	12	9	23.20715 seg

Tabla 3: Tecnología I y II.

desempeño del algoritmo basado en Búsqueda Armónica sobresale al observar que las mejores soluciones reportadas en [7] requieren de 13 trimestres para poder acomodar las UEA sin incurrir en violaciones, mientras que en este trabajo se encontraron soluciones que solamente requiere de 12 y 11 trimestres.

De esta forma se concluye que el desempeño del algoritmo propuesto ha mostrado ser capaz de superar el desempeño de las técnicas reportadas hasta este momento, en un conjunto pequeño de instancias reales. Es importante destacar que se pueden hacer mejoras que ayuden a disminuir el número de soluciones no válidas devueltas por el algoritmo, mediante la hibridación de diferentes técnicas de búsqueda adecuadas al problema de currículo académico, pero dicho trabajo se encuentra más allá de la propuesta realizada en este proyecto de integración.

8 CONCLUSIONES

En este proyecto se implementó un algoritmo basado en la técnica heurística Búsqueda Armónica para resolver un problema de empaquetamiento con restricciones de precedencia, para lo cual se empleó información real obtenida de los planes de estudio de las carreras de Ingeniería en Computación y de Ingeniería Física, de la Universidad Autónoma Metropolitana Unidad Azcapotzalco. Basado en los resultados obtenidos con las instancias creadas, se puede decir que el algoritmo propuesto es capaz de generar soluciones acordes a las restricciones requeridas por la universidad.

Al comparar las soluciones obtenidas con los resultados reportados en el proyecto de integración [7], se observó que BA superó el desempeño de Optimización por Enjambre de Partículas, al ser capaz de generar soluciones válidas con un menor número de trimestres en las 4 instancias de Ingeniería Física.

Finalmente, se considera que el algoritmo propuesto en este trabajo puede ser mejorado si se incluyen estrategias de búsqueda adecuadas al problema de currículo académico, que ayuden a aumentar el número de soluciones válidas obtenidas. Esto puede incluir el análisis y desarrollo de técnicas específicas o bien la hibridación con otro tipo de técnicas heurísticas. Sin embargo, esto se encuentra fuera de los alcances de este proyecto de integración, y se deja como una futura línea de investigación.

Parte I

ANEXOS

A

INSTANCIAS INGENIERÍA EN COMPUTACIÓN.

Clave[1] Créditos[2] Seriación 1[3] Seriación 2[4] Seriación 3[5] Seriación 4[6]

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3
1100033	3			
1111078	4			
1112026	7			
1201008	4			
1111079	9	1111078	1112026	
1111092	3	1111079		
1111081	9	1111079		
1111093	3	1111081	1111092	
1111083	9	1111081	1112029	
1112013	9	1112026		
1112027	6	1112026		
1112028	9	1112027		
1112029	9	1112028		
1112030	9	1112029		
1113046	6	1112028	1111081	
1113084	9			
1113085	3	1113086		
1113086	6	1113084		
1113087	3	1113085	1113086	
1151038	7	1112013	1112027	
1151039	7	1151038	1112029	
1153001	9	1112029		
1112017	9	1151038		
1112033	9	1151038		
1112034	9	1112033		
1112040	9	1112030		

Tabla 4: Obligatorias de Ingeniería en Computación.

1 Identificador que se le asigna a una UEA
2 Unidad que mide el tiempo de formación de un estudiante
3 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
4 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
5 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
6 Clave de la UEA pre-requisito para inscribir la UEA en columna clave

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3
1121025	9	1121060		
1121033	3	1121040	1121060	
1121060	9	1121037	1151042	
1121037	12	1151038		
1121038	9	1112040	1121060	
1121040	6	1121037		
1121043	12	1121038		
1124052	9	1111083	1113086	
1151018	9	1121025	1121038	
1151040	9	1152001	1151042	1112033
1154041	8	1151042	1153001	
1151042	8	1151038		
1151044	8	1151038		
1151046	9	1151018	1151047	
1151047	12	1151072	1151041	
1151048	8	1151072	1151041	
1151049	9	1121060	1112034	
1151050	6	1100040		
1151051	9	1151042	1112017	
1151052	7	1151072		
1151072	3	1151044		
1152001	9	1151039		
1100037	6	50 Créditos		
1100038	6	1100037		
1100040	6	280 Créditos		
1100039	6	1100040		
1100041	6	1100039		
1100103	3	360 Créditos	1100039	
1100113	18	1100103		

Tabla 5: Obligatorias de Ingeniería en Computación.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3	Seriacion 4
1112022	6	1151038			
1112035	9	1151040			
1112037	9	1112034			
1112038	9	1112034			
1151045	9	1151062			
1151061	6	1151040			
1151062	9	1151042	1153001		
1151063	9	1151042	1153001		
1151064	3	1151061	1151062	1151063	1112035
1151065	9	1112017	1151040		
1151066	9	1151040	1112017		
1151067	9	1151040			
1151068	9	1151051			
1152002	9	1152001	1153001		

Tabla 6: Área de Concentración de Algoritmos e Inteligencia Artificial.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3
1100034	6	1151063	1121032	1124043
1100035	9	1100034		
1121032	3	1121034		
1121034	9	1121060	1124052	
1123040	9	1124001	1124005	
1123041	9	1123040	1123045	
1123043	9	1121034	1123041	1123046
1123045	3	1123040		
1123046	3	1123041		
1123048	3	1123043		
1124001	9	1112030		

Tabla 7: Área de Concentración Mecatrónica.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3	Seriacion 4
1151054	7	1151048	1151047	1121038	
1151055	8	1151048	1151047		
1151056	8	1151048			
1151057	6	1151048	1151047	1121038	
1151058	7	1151054	1151056	1151057	1151074
1151059	7	1151054			
1151060	9	1151044			
1151071	9	1151048			
1151074	8	1151046	1151047		
1112005	12	1112029	1112013		

Tabla 8: Área de Concentración Sistemas de Información.

B

INSTANCIAS INGENIERÍA FÍSICA

Clave[1] Créditos[2] Seriación 1[3] Seriación 2[4] Seriación 3[5] Seriación 4[6]

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3	Seriacion 4
1100033	3				
1111078	4				
1112026	7				
1201008	4				
1111079	9	1111079	1111078		
1111092	3	1111092	1111079		
1111081	9	1111081	1111079		
1111093	3	1111093	1111081	1111092	
1111083	9	1111083	1111081		
1112013	9	1112013	1112026		
1112027	6	1112027	1112026		
1112028	9	1112028	1112027		
1112029	9	1112029	1112028		
1112030	9	1112030	1112029		
1113046	6	1113046	1112028	1111081	
1113084	9				
1113085	3				
1113086	6	1113086	1113084		
1113087	3	1113087	1113085		
1151038	7	1151038	1112013	1112027	
1151039	7	1151039	1151038		
1153001	9	1153001	1112029		
1111013	9	1111013	1111081	1112005	1112030
1111019	9	1111019	1137007		
1111043	9	1111043	1111090	1111091	
1111044	9	1111044	1111043		

Tabla 9: Obligatorias de Ingeniería Física.

1 Identificador que se le asigna a una UEA
2 Unidad que mide el tiempo de formación de un estudiante
3 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
4 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
5 Clave de la UEA pre-requisito para inscribir la UEA en columna clave
6 Clave de la UEA pre-requisito para inscribir la UEA en columna clave

Clave	Créditos	Seriacion 1	Seriacion 2	Seriacion 3	Seriacion 4
1111048	9	1111048	1111090	1111091	
1111055	9	1111055	1111090		
1111069	3				
1111085	9	1111085	1112029	1112005	
1111087	3				
1111088	3	1111088	1111048		
1111094	3				
1111090	9	1111090	1111083	1112030	
1111091	9	1111091	1111085	1112015	
1112005	12	1112005	1112029	1112013	
1112015	9	1112015	1112030		
1112016	6	1112016	1112005		
1113069	9	1113069	1113046		
1113070	3				
1122012	9	1122012	1112015		
1124001	9				
1124005	3				
1132064	3	1132064	1133048		
1133048	6	1133048	1153001		
1137005	9	1137005	1111081	1112030	
1137006	9	1137006	1113046		
1137007	9	1137007	1112029	1137006	
1154029	9	1154029	1153001		
1100037	6	50 Créditos			
1100038	6	1100038	1100037		
1100040	6	280 Créditos			
1100039	6	1100039	1100040		
1100041	6	1100041	1100039		
1100106	3	360 Créditos	1100106	1100039	
1100116	18	1100116	1100106		

Tabla 10: Obligatorias de Ingeniería Física.

Clave	Créditos	Seriacion 1	Seriacion 2
1100077	6	150 Créditos	
1100078	6	150 Créditos	
1100079	6	150 Créditos	
1100080	6	150 Créditos	
1111054	9	1123040	
1111058	9	1123016	
1111060	9	1123016	
1121037	12	1151038	
1121040	6		
1123016	9	1123040	
1123021	9	1121037	
1123045	3		
1123040	9	1124001	1124005

Tabla 11: Área de Concentración Instrumentación. Instancia I.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriación 3
1100077	6	150 Créditos		
1100078	6	150 Créditos		
1100079	6	150 Créditos		
1100080	6	150 Créditos		
1123040	9	1124001	1124005	
1123041	9	1123040	1123045	
1123042	9	1123043	1123021	
1123043	9	1123021	1123041	1123046
1123044	9	1123041	1124003	
1123045	3			
1123021	9	1121037		
1124003	9	1124001	1112015	
1123046	3			
1121037	12	1151038		

Tabla 12: Área de Concentración Instrumentación. Instancia II.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriación 3	Seriación 4
1100077	6	150 Créditos			
1100078	6	150 Créditos			
1100079	6	150 Créditos			
1100080	6	150 Créditos			
1111032	9	1111048			
1111034	9	1111043			
1133014	9	1145054			
1141006	3				
1145052	6	1145054			
1145054	9	1112028	1113086	1113087	1113046
1145056	9	1112030			
1145057	3				
1145071	6	1145054			
1145072	3	1145052			

Tabla 13: Área de Concentración Tecnología. Instancia I.

Clave	Créditos	Seriacion 1	Seriacion 2	Seriación 3	Seriación 4
1100077	6	150 Créditos	1111048		
1100078	6	150 Créditos	1111081	1112005	
1100079	6	150 Créditos			
1100080	6	150 Créditos	1145054		
1111032	9	1111032	1111032		
1111045	9	1111045	1145054		
1142025	3	1112028	1113086	1113087	1113046
1145001	6	1145001	1137006		
1145050	6	1145050	1145052		
1145052	6	1145052	1145060		
1145054	9	1145054	1145054		
1145058	9	1145058			
1145060	9	1145060			
1145066	9	1145066			
1146038	9	1146038			

Tabla 14: Área de Concentración Tecnología. Instancia II.

C

RESULTADOS PARA INGENIERÍA EN COMPUTACIÓN. PRIMERA PARTE

Instancia[1] Trimestres Empleados[2] Factible[3] MaxExcedidos[4] MaxMinimos[5] MaxSeriacion[6] Tiempo de Ejecución[7]

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	Si	0	0	0	20.9510 seg
Mecatrónica	12	Si	0	0	0	18.7140 seg
Sistemas de Información	13	Si	0	0	0	19.2560 seg

Tabla 15: Resultados con la semilla 83554.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	21.6740 seg
Mecatrónica	12	Si	0	0	0	21.6660 seg
Sistemas de Información	12	Si	0	0	0	20.7020 seg

Tabla 16: Resultados con la semilla 51025.

1 Área de Concentración de Ingeniería en Computación
2 Número de trimestres en el que puedes terminar la licenciatura
3 Se considera factible si el número máximo de trimestres empleados es 13
4 Número de trimestres en los cuales nos hemos pasado de los créditos permitidos
5 Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas
6 Número de UEA que no cumplieron con la seriación
7 Tiempo que tarda en generar una solución

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	0	19.3260 seg
Mecatrónica	12	Si	0	0	0	19.5160 seg
Sistemas de Información	13	No	0	0	1	21.9380 seg

Tabla 17: Resultados con la semilla 63896.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	Si	0	0	0	21.0160 seg
Mecatrónica	12	No	0	0	1	18.9470 seg
Sistemas de Información	12	Si	0	0	0	21.1480 seg

Tabla 18: Resultados con la semilla 34135.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	Si	0	0	0	20.6700 seg
Mecatrónica	11	No	4	0	1	18.3190 seg
Sistemas de Información	13	Si	0	0	0	21.1250 seg

Tabla 19: Resultados con la semilla 26575.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	14	No	0	0	0	19.9360 seg
Mecatrónica	12	Si	0	0	0	18.5030 seg
Sistemas de Información	12	No	4	0	1	21.8490 seg

Tabla 20: Resultados con la semilla 74697.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	No	2	0	0	19.8170 seg
Mecatrónica	12	Si	0	0	0	20.7270 seg
Sistemas de Información	12	No	1	0	0	20.3040 seg

Tabla 21: Resultados con la semilla 52563.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	14	No	0	0	0	21.6410 seg
Mecatrónica	12	Si	0	0	0	19.0370 seg
Sistemas de Información	12	No	3	0	1	21.2260 seg

Tabla 22: Resultados con la semilla 33729.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	No	2	0	0	22.2890 seg
Mecatrónica	12	Si	0	0	0	23.2060 seg
Sistemas de Información	13	No	1	0	1	20.3950 seg

Tabla 23: Resultados con la semilla 83848.

D

RESULTADOS PARA INGENIERÍA EN COMPUTACIÓN. SEGUNDA PARTE

Instancia[1] Trimestres Empleados[2] Factible[3] MaxExcedidos[4] MaxMinimos[5] MaxSeriacion[6] Tiempo de Ejecución[7]

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	0	20.3220 seg
Mecatrónica	12	Si	0	0	0	22.7600 seg
Sistemas de Información	13	No	0	0	1	20.6530 seg

Tabla 24: Resultados con la semilla 44431.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	21	No	5	0	0	19.1740 seg
Mecatrónica	13	No	0	0	2	19.0930 seg
Sistemas de Información	12	No	6	0	0	21.8350 seg

Tabla 25: Resultados con la semilla 48767.

1 Área de Concentración de Ingeniería en Computación
2 Número de trimestres en el que puedes terminar la licenciatura
3 Se considera factible si el número máximo de trimestres empleados es 13
4 Número de trimestres en los cuales nos hemos pasado de los créditos permitidos
5 Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas
6 Número de UEA que no cumplieron con la seriación
7 Tiempo que tarda en generar una solución

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	14	No	0	0	0	20.3260 seg
Mecatrónica	12	Si	0	0	0	19.4700 seg
Sistemas de Información	12	Si	0	0	0	19.8650 seg

Tabla 26: Resultados con la semilla 91416.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	10	0	0	19.9730 seg
Mecatrónica	12	No	3	0	0	19.7430 seg
Sistemas de Información	12	No	1	0	1	22.0320 seg

Tabla 27: Resultados con la semilla 35462.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	Si	0	0	0	19.7780 seg
Mecatrónica	13	No	0	0	1	19.6050 seg
Sistemas de Información	17	No	0	0	2	20.0450 seg

Tabla 28: Resultados con la semilla 49700.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	2	19.4590 seg
Mecatrónica	11	No	0	0	1	19.1480 seg
Sistemas de Información	13	No	2	0	1	19.8350 seg

Tabla 29: Resultados con la semilla 44077.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	Si	0	0	0	19.6890 seg
Mecatrónica	11	No	0	0	2	18.6800 seg
Sistemas de Información	13	Si	0	0	0	19.4860 seg

Tabla 30: Resultados con la semilla 7007.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	Si	0	0	0	19.5860 seg
Mecatrónica	19	No	0	0	2	19.3820 seg
Sistemas de Información	12	No	6	0	0	21.2090 seg

Tabla 31: Resultados con la semilla 60000.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	6	0	0	19.7650 seg
Mecatrónica	19	No	0	0	2	18.6920 seg
Sistemas de Información	18	No	1	0	0	19.9290 seg

Tabla 32: Resultados con la semilla 21585.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	No	5	0	0	20.7910 seg
Mecatrónica	11	No	1	0	1	19.9380 seg
Sistemas de Información	12	No	3	0	0	20.3660 seg

Tabla 33: Resultados con la semilla 94543.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	13	No	2	0	0	19.7730 seg
Mecatrónica	12	No	2	0	0	20.0770 seg
Sistemas de Información	12	No	3	0	1	20.1020 seg

Tabla 34: Resultados con la semilla 81553.

E

RESULTADOS PARA INGENIERÍA FÍSICA. PRIMERA PARTE

Instancia[1] Trimestres Empleados[2] Factible[3] MaxExcedidos[4] MaxMinimos[5] MaxSeriacion[6] Tiempo de Ejecución[7]

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	17.2550 seg
Instrumentación II	16	No	0	0	0	23.2680 seg
Tecnología I	12	Si	0	0	0	18.8720 seg
Tecnología II	13	Si	0	0	0	24.4620 seg

Tabla 35: Resultados con la semilla 83554.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	12	Si	0	0	0	22.3350 seg
Instrumentación II	11	No	4	0	0	22.9630 seg
Tecnología I	14	No	0	0	0	18.3690 seg
Tecnología II	12	No	0	0	1	23.2130 seg

Tabla 36: Resultados con la semilla 51025.

1 Datos de Ingeniería Física
2 Número de trimestres en el que puedes terminar la licenciatura
3 Se considera factible si el número máximo de trimestres empleados es 13
4 Número de trimestres en los cuales nos hemos pasado de los créditos permitidos
5 Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas
6 Número de UEA que no cumplieron con la seriación
7 Tiempo que tarda en generar una solución

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	No	1	0	0	19.6760 seg
Instrumentación II	12	Si	0	0	0	22.3010 seg
Tecnología I	12	Si	0	0	0	18.3540 seg
Tecnología II	12	No	0	0	1	23.0610 seg

Tabla 37: Resultados con la semilla 63896.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	3	0	0	22.2100 seg
Instrumentación II	14	No	0	0	0	21.9650 seg
Tecnología I	11	No	2	0	0	18.3680 seg
Tecnología II	12	Si	0	0	0	23.0540 seg

Tabla 38: Resultados con la semilla 34135.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	19.2510 seg
Instrumentación II	15	No	2	0	2	21.7780 seg
Tecnología I	11	No	4	0	0	23.6300 seg
Tecnología II	12	Si	0	0	0	22.6410 seg

Tabla 39: Resultados con la semilla 26575.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	5	0	0	18.8460 seg
Instrumentación II	12	Si	0	0	0	24.4860 seg
Tecnología I	11	No	2	0	1	24.4940 seg
Tecnología II	12	No	4	0	1	23.5060 seg

Tabla 40: Resultados con la semilla 74697.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	19.2110 seg
Instrumentación II	11	No	2	0	0	22.4350 seg
Tecnología I	12	Si	0	0	0	18.6400 seg
Tecnología II	12	No	0	0	1	23.3350 seg

Tabla 41: Resultados con la semilla 52563.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	19.8280 seg
Instrumentación II	11	No	0	0	1	22.5650 seg
Tecnología I	12	No	0	0	1	18.3760 seg
Tecnología II	13	Si	0	0	0	22.9450 seg

Tabla 42: Resultado con las semillas 33729.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	12	No	4	0	0	18.5260 seg
Instrumentación II	12	No	0	0	1	21.7580 seg
Tecnología I	11	No	6	0	0	18.3370 seg
Tecnología II	14	No	0	0	0	22.8680 seg

Tabla 43: Resultados con la semilla 83848.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	18.6510 seg
Instrumentación II	13	Si	0	0	0	22.0330 seg
Tecnología I	11	No	4	0	0	17.9730 seg
Tecnología II	12	Si	0	0	0	23.0660 seg

Tabla 44: Resultados con la semilla 44431.

F

RESULTADOS PARA INGENIERÍA FÍSICA. SEGUNDA PARTE

Instancia[1] Trimestres Empleados[2] Factible[3] MaxExcedidos[4] MaxMinimos[5] MaxSeriacion[6] Tiempo de Ejecución[7]

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	19.0140 seg
Instrumentación II	12	Si	0	0	0	22.1500 seg
Tecnología I	13	Si	0	0	0	16.8130 seg
Tecnología II	14	No	2	0	0	23.8460 seg

Tabla 45: Resultados con la semilla 48767.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	12	Si	0	0	0	18.8360 seg
Instrumentación II	16	No	0	0	0	18.5010 seg
Tecnología I	12	No	2	0	0	18.6520 seg
Tecnología II	11	No	5	0	0	22.6100 seg

Tabla 46: Resultados con la semilla 91416.

1 Datos de Ingeniería Física
2 Número de trimestres en el que puedes terminar la licenciatura
3 Se considera factible si el número máximo de trimestres empleados es 13
4 Número de trimestres en los cuales nos hemos pasado de los créditos permitidos
5 Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas
6 Número de UEA que no cumplieron con la seriación
7 Tiempo que tarda en generar una solución

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	18.7440 seg
Instrumentación II	11	No	2	0	0	18.0470 seg
Tecnología I	11	No	5	0	0	21.5210 seg
Tecnología II	12	Si	0	0	0	22.4950 seg

Tabla 47: Resultados con la semilla 35462.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	12	Si	0	0	0	18.2490 seg
Instrumentación II	11	Si	0	0	0	19.0330 seg
Tecnología I	11	No	4	0	0	21.5860 seg
Tecnología II	12	Si	0	0	0	22.8930 seg

Tabla 48: Resultados con la semilla 49700.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	19.5380 seg
Instrumentación II	17	No	0	0	0	19.9690 seg
Tecnología I	11	No	2	0	1	23.7040 seg
Tecnología II	12	Si	0	0	0	23.2180 seg

Tabla 49: Resultados con la semilla 44077.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	No	2	0	0	18.9650 seg
Instrumentación II	12	No	2	0	0	18.7470 seg
Tecnología I	12	No	2	0	0	24.6460 seg
Tecnología II	16	No	0	0	2	22.8780 seg

Tabla 50: Resultados con la semilla 7007.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	17.7360 seg
Instrumentación II	12	No	0	3	0	19.9100 seg
Tecnología I	11	No	4	0	1	23.5160 seg
Tecnología II	12	Si	0	0	0	23.6370 seg

Tabla 51: Resultados con la semilla 60000.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	No	4	0	0	17.7020 seg
Instrumentación II	11	No	2	0	0	17.2580 seg
Tecnología I	12	Si	0	0	0	25.0160 seg
Tecnología II	12	Si	0	0	0	23.1620 seg

Tabla 52: Resultados con la semilla 21585.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	Si	0	0	0	17.2940 seg
Instrumentación II	12	Si	0	0	0	18.2150 seg
Tecnología I	12	Si	0	0	0	24.3090 seg
Tecnología II	12	Si	0	0	0	24.0190 seg

Tabla 53: Resultados con la semilla 94543.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	11	No	2	0	2	17.9200 seg
Instrumentación II	16	No	0	0	0	22.6970 seg
Tecnología I	13	No	0	0	1	25.3320 seg
Tecnología II	18	No	0	0	1	23.2340 seg

Tabla 54: Resultados con la semilla 81553.

G

CODIGO FUENTE

En las siguientes páginas se muestra un listado con el código fuente desarrollado.

G.1 SOLUCION.CPP

```
#include "Solucion.h"
#include <iostream>
#include <algorithm>
#include <stdio.h>

long Semilla = 81553;
int UEAsMaxTrim, MaxTrim;

int Solucion::maximoTrimestre(){ //funcion que regresa un entero el cual sera el numero maximo que representa el trimestre
                int max; //variable max de tipo entero que contendra el trimestre maximo
                if(trimestreAleatorio.size()==0){
                   cout << "Error, no hay elementos" << endl;
                   }
                   else{

                       max=trimestreAleatorio[0]; //el maximo  sera igual al trimestreAleatorio en la posicion cero si se encuentra en esa posicion
                       for(int i=0; i<trimestreAleatorio.size(); i++){// for para recorrer la lista de los trimestre aleatorios
                           if (trimestreAleatorio[i] > max)
                                   max=trimestreAleatorio[i];
                                       }
                       UEAsMaxTrim = 0;
                       for(int i=0; i<trimestreAleatorio.size(); i++){
                           if (trimestreAleatorio[i] == max)
                                   UEAsMaxTrim++;
                                       }
                                       }

                                       return max;  //regresa el trimestre maximo
                                       }

        int Solucion::SiguienteAleatorioEnteroModN(long * semilla, int n) // Funcion que genera un numero aleatorio entre 0 y n
                {
                    double a;
                    int v;
                    long double zi, mhi31 = 2147483648u, ahi31 = 314159269u, chi31 = 453806245u;
                    long int dhi31;
                    zi = *semilla;
                    zi = (ahi31 * zi) + chi31;
                    if(zi > mhi31)
                    {
                        dhi31 = (long int) (zi / mhi31);
                        zi = zi - (dhi31 * mhi31);
                    }
                    *semilla = (long int)zi;
                    zi = zi / mhi31;
                    a = zi;
                    v = (int)(a * (n+1));
                    if (v == (n+1))
                        return(v-1);
                    return(v);

                }

        double SiguienteAleatorioRealoy1(long * semilla)
        //DEVUELVE UN ALEATORIO ENTRE 0 Y 1
        {
            long double zi, mhi31 = 2147483648u, ahi31 = 314159269u, chi31 = 453806245u;
            long int dhi31;
            zi = *semilla;
            zi = (ahi31 * zi) + chi31;
            if(zi > mhi31)
            {
                dhi31 = (long int) (zi / mhi31);
                zi = zi - (dhi31 * mhi31);
            }
            *semilla = (int) zi;
```

```
67                 zi = zi / mhi31;
68                 return (zi);
69             }
70
71             void Solucion::crearTrimestre(int nTrimestre){ //Funcion  que agrega a
                   la lista de trimestres Aleatorios un numero aleatorio que genera
                   otra funcion
72
73                 for(int i=0; i<nTrimestre; i++){
74                         trimestreAleatorio.push_back(SiguienteAleatorioEnteroModN(&
                               Semilla, numeroTrimestres-2)+1);
75                         }
76             }
77
78             void Solucion::funcionObjetivo(vector<UEA>listaDatos, vector<UEA>
                   listaCreditosMinimos, vector<Seriacion> listaSeriacion){//esta
                   funcion es la que va a calcular el costo de la solucion
79                 maxTrim = maximoTrimestre();
80                 maxExcedidos= violacionesPorCreditosExcedidos(listaDatos);
81                 maxMinimos = violacionesCreditosMinimos(listaDatos,
                       listaCreditosMinimos);
82                 maxSeriacion = ViolacionesSeriacion(listaDatos, listaSeriacion);
83                 costo = 1 * maxTrim + 1 * maxExcedidos  + 1 * maxMinimos + 4 *
                       maxSeriacion + (30 * maxTrim + 5 * UEAsMaxTrim);
84                 }
85
86
87             Solucion::Solucion(vector<UEA> listaDatos, vector<UEA>
                   listaCreditosMinimos, vector<Seriacion> listaSeriacion){ //
                   Constructor numeroUEAs es el tamano del vector
88                     crearTrimestre(listaDatos.size()); // funcion que recibe el
                           numero de UEAs  para asignarle a cada una un numero
                           aleatorio que correspondera al trimestre
89                     //costo=funcionObjetivo();  //  el costo vendra del total que
                           arroje la funcion objetivo
90                     for(int i=0; i<numeroTrimestres; i++){
91                      creditosTrimestre[i]=0;
92                      }
93                      llenaCreditosTrimestre(listaDatos);
94                      funcionObjetivo(listaDatos, listaCreditosMinimos,
                           listaSeriacion);
95
96                       }
97
98             Solucion::Solucion(vector<Solucion> listaSoluciones, vector<UEA>
                   listaDatos, vector<UEA> listaCreditosMinimos, vector<Seriacion>
                   listaSeriacion){
99
100
101                int i=0, k;
102                float b1, b2, b3, b4;
103
104
105                b3 = SiguienteAleatorioRealoy1(& Semilla);
106                if(b3 < 0.5)
107                 b4 = 0.01;
108                if(0.5 <= b3)
109                 b4 = 0.99;
110
111                i= SiguienteAleatorioEnteroModN(& Semilla, listaSoluciones.size()
                       -1); //se asigna un valor aleatorio al indice entre 1 y el
                       numero de usuario
112
113                for(int j=0; j<listaSoluciones[0].tamanoTriAleatorio(); j++){ // se
                       crea la nueva Solucion con partes de las anteriores
114
115                    b3 = SiguienteAleatorioRealoy1(& Semilla);
116                    if(b3 < 0.2)
117                        i= SiguienteAleatorioEnteroModN(& Semilla,
                               listaSoluciones.size()-1); //se asigna un valor
                               aleatorio al indice entre 1 y el numero de usuario
118                    //cout << i << endl;
119                    //trimestreAleatorio.push_back(listaSoluciones[i].
                           posTriAleatorio(j));
120                    b1 = SiguienteAleatorioRealoy1(& Semilla);
121                    if(b1 <= 0.97){
122                        b2 = SiguienteAleatorioRealoy1(& Semilla);
123                        if(b2 <= 0.9){
124                                trimestreAleatorio.push_back(listaSoluciones[
                                       i].posTriAleatorio(j));
125                        }
126                        else{
127                            //k = SiguienteAleatorioEnteroModN(& Semilla, 1) +
                                   1;//aleatorio entre 1 y 2
128                            k = 1;
129                            b3 = SiguienteAleatorioRealoy1(& Semilla);
130                            if(b3 <= b4){
131                                if(listaSoluciones[i].posTriAleatorio(j) + k <=
                                       RangoMax[j]) //TrimestreMaximo)
132                                trimestreAleatorio.push_back(listaSoluciones[
                                       i].posTriAleatorio(j) + k);
133                            else{
134                                trimestreAleatorio.push_back(RangoMax[j] - k)
                                       ; //listaSoluciones[i].posTriAleatorio(
                                       j) - k);
135                            }
```

```
136                                     //trimestreAleatorio.push_back(
                                            TrimestreMaximo);
137                                 }
138                                 else{
139                                     if(RangoMin[j] <= listaSoluciones[i].
                                            posTriAleatorio(j) - k)
140                                         trimestreAleatorio.push_back(listaSoluciones[
                                                i].posTriAleatorio(j) - k);
141                                     else
142                                         trimestreAleatorio.push_back(RangoMin[j]);//
                                                1);
143                                 }
144                             }
145                         }
146                         else{
147                             //k = SiguienteAleatorioEnteroModN(& Semilla,
                                    TrimestreMaximo - 1) + 1;//aleatorio entre 1 y 2
148                             k = RangoMax[j] - RangoMin[j];
149                             if(k > 0)
150                                 k = SiguienteAleatorioEnteroModN(& Semilla, k) +
                                        RangoMin[j];//aleatorio entre 1 y 2
151                             else
152                                 k = RangoMin[j];
153                             //cout << k << "  " << j << "\n";
154                             //getchar();
155                             trimestreAleatorio.push_back(k);
156                             //trimestreAleatorio.push_back(
                                    SiguienteAleatorioEnteroModN(& Semilla,
                                    TrimestreMaximo-1) + 1);
157                           //  cout << " ... " << trimestreAleatorio[j] << "\n";
158                         }
159
160
161
162                     }
163
164                 for(int i=0; i<numeroTrimestres; i++){ //asignamos creditos a los
                        trimestres de la nueva solucion
165                 creditosTrimestre[i]=0;
166                   }
167                   llenaCreditosTrimestre(listaDatos);
168                   funcionObjetivo(listaDatos, listaCreditosMinimos, listaSeriacion)
                          ;
169
170
171                                             }
172
173             int Solucion::tamanoTriAleatorio(){
174                 return trimestreAleatorio.size();
175             }
176
177             int Solucion::posTriAleatorio(int j){
178
179                 return trimestreAleatorio[j];
180             }
181
182
183             int Solucion::getCosto(){
184                 //cout << "Costo total: " << costo << endl;
185                 return costo;
186                         }
187
188             void Solucion::llenaCreditosTrimestre (vector<UEA> listaDatos) {
189
190                  int trimestre;
191                  for(int i=0; i<trimestreAleatorio.size(); i++){
192                          trimestre=trimestreAleatorio[i];
193                          creditosTrimestre[trimestre] += listaDatos[i].getCreditos
                                ();
194
195                  }
196
197             int Solucion::violacionesPorCreditosExcedidos(vector<UEA> listaDatos){
                   //funcion que verifica si nos pasamos de creditos en cada
                   trimestre
198
199                   int max=0;
200
201                   for(int i=1; i<numeroTrimestres; i++){
202
203                          if(creditosTrimestre[i]>maxCreditos){
204                          max+=creditosTrimestre[i]-maxCreditos; // entonces ese
                                numero de trimestre aleatorio sera igual al maximo
205                                                     }
206                     }
207                                 return max;  //regresa el maximo
208
209              }
210
211 int Solucion::violacionesCreditosMinimos(vector<UEA> listaDatos, vector<UEA>
        listaCreditosMinimos ){
212                 int creditosAcumulados[numeroTrimestres]; //arreglo que contiene el
                        numero de creditos que se van acumulando en cada trimestre
213                 int max=0;
214                 int claveActual;
215                 int j=0;
```

```
216                 int creditosNecesarios;
217                 int credTrimAnterior;
218                 int trimAnterior;
219
220
221                 creditosAcumulados[0]= 0;
222                 for(int i=1; i<numeroTrimestres; i++){ //con este for llenamos  el
                          arreglo creditosAcumulados para obtener el total de creditos
                          por trimestre
223                 creditosAcumulados[i] = creditosTrimestre[i]+creditosAcumulados[i
                          -1];
224                   }
225
226                 for(int i=0; i<listaCreditosMinimos.size(); i++){ //con este for
                          recorremos la lista de creditosMinimos
227                   claveActual=listaCreditosMinimos[i].getClave();
228                   while(j<listaDatos.size() && listaDatos[j].getClave() !=
                              claveActual){//este while es para avanzar en la
                              listaDatos  cuando la claveActual no es igual a la que
                              buscamos
229                       j++;
230                   }
231
232                   trimAnterior=trimestreAleatorio[j]-1;
233                   creditosNecesarios=listaCreditosMinimos[i].getCreditos();
234                   credTrimAnterior=creditosAcumulados[trimAnterior];
235
236                   if( (creditosNecesarios-credTrimAnterior) > 0){//con este if
                            obtenemos la diferencia de creditos que hacen falta para
                            cursa la uea
237                     }
238
239                   if( (creditosNecesarios-credTrimAnterior) > 0){
240                     max+=creditosNecesarios-credTrimAnterior;
241                     }
242                 }
243
244                  return (max);
245             }
246
247
248         int Solucion::ViolacionesSeriacion(vector<UEA> listaDatos, vector<Seriacion>
                  listaSeriacion ){
249
250             int max=0;
251             int claveMax=0;
252             int claveSeriacion;
253             int j=0;
254             int malas;
255             vector<UEA>::iterator posRelativa;
256             int indice;
257             int trimAnterior;
258             int trimActual;
259
260
261             for(int i=0; i<listaSeriacion.size(); i++){
262                     claveSeriacion=listaSeriacion[i].claves[0];
263                     while(j<listaDatos.size() && listaDatos[j].getClave() !=
                              claveSeriacion){
264                       j++;
265                     }
266
267                     malas=0;
268
269                     for(int k=1; k<listaSeriacion[i].claves.size(); k++){
270
271                     UEA elemBuscar;
272                     elemBuscar.setClave(listaSeriacion[i].claves[k]);
273
274                     //lower_bound es la funcion que hace la Busqueda binaria en
275                    //un vector ordenado. Lo que nos regresa es un iterador, que
276                    //es una especie de apuntador a la posicion donde encontro el
277                    //elemento buscado.
278                     posRelativa = lower_bound(listaDatos.begin(), listaDatos.end(),
                              elemBuscar);
279
280                    //con esta operacion obtenemos el indice real del objeto que
                            buscamos
281                   indice = posRelativa - listaDatos.begin();
282
283                   trimAnterior=trimestreAleatorio[indice];
284                   trimActual=trimestreAleatorio[j];
285
286                   if(RangoMax[indice] > TrimestreMaximo)
287                       RangoMax[indice] = TrimestreMaximo;
288                   if(RangoMax[j] > TrimestreMaximo)
289                       RangoMax[j] = TrimestreMaximo;
290
291                   if(RangoMax[indice] >= RangoMax[j])
292                       RangoMax[indice] = RangoMax[j] - 1;
293                   if(RangoMin[j] <= RangoMin[indice])
294                       RangoMin[j] = RangoMin[indice] + 1;
295
296                   if(trimAnterior >= trimActual ){
297                           malas += trimAnterior - trimActual + 1;
298                     }
```

```
299                 }
300
301                                 max += malas;
302                                 claveMax= listaSeriacion[i].claves[0];
303
304               }
305
306             return (max);
307           }
308
309
310            void Solucion::imprime(){ //funcion que imprime la solucion en pantalla,
                    es decir cada trimestre aleatorio
311                         cout << "(" ;
312                         for(int i=0; i<trimestreAleatorio.size(); i++){//for para
                              ir recorriendo toda la lista que contiene los
                              trimestres aleatorios
313                         if(trimestreAleatorio[i]<10)
314                         cout << "0";
315                         cout << trimestreAleatorio[i] << ", ";
316                            }
317                         cout << ")\n" << endl;
318            }
319
320
321            void Solucion::imprimeCostos(int sol){
322
323                   if (sol==79999){
324                         cout << "Max Trim: " << maxTrim << " Max Excedidos: " <<
                              maxExcedidos << " Max Minimos: " << maxMinimos << "
                              Max Seriacion: " << maxSeriacion <<  endl;
325                         cout << "Costo total: " << costo << endl;
326                         }
327                         if(TrimestreMaximo > maxTrim)
328                               TrimestreMaximo = maxTrim;
329
330            }
331
332
333           trimPos::trimPos(int a, int b){ //constructora
334                       nTrim=a;
335                       indice=b;
336           }
337
338
339          int trimPos::operator<(const trimPos &otroObjeto) const{
340
341               if (nTrim < otroObjeto.nTrim)
342                  return 1;
343               return 0;
344           }
345
346
347           void Solucion::imprime2(vector<UEA> listaDatos){
348
349                         int a;
350                         int b;
351                         int clave;
352                         int c;
353
354                         vector<trimPos> trimestrePos;
355                         for(int i=0; i<trimestreAleatorio.size(); i++){
356                             a= trimestreAleatorio[i];
357                             b= i;
358                             trimPos Objeto(a, b);
359                             trimestrePos.push_back(Objeto);
360                         }
361
362                         sort(trimestrePos.begin(), trimestrePos.end());
363
364                         //Se crea una variable auxiliar, la cual se inicializa en
                              1
365                         //para indicar  que empieza en el trimestre 1
366                         int aux = 1;
367
368                         for (int i=0; i<trimestrePos.size(); i++) {
369                              if (trimestrePos[i].nTrim == aux){
370                                   cout << "\nTrimestre " << trimestrePos[i].nTrim
                                        << ": ";
371                                   aux++;
372                                   }
373
374                              clave = listaDatos[trimestrePos[i].indice].getClave()
                                   ;
375                              cout << clave << ", ";
376                         }
377
378
379           }
```

Listado 1: Solucion.cpp

G.2 SOLUCION.H

```
#include <vector>
#include "UEA.h"
#include "Seriacion.h"

using namespace std;
const int numeroTrimestres=37; //uno mas que los posibles trimestres a cursar
const int maxCreditos=52;
extern int TrimestreMaximo;
extern int RangoMin[100], RangoMax[100];


class Solucion{
      private:
              int costo; //creamos variable costo de tipo entero, la cual tendra el
                    valor del costo de cada solucion
              vector<int> trimestreAleatorio; //lista soluciones que contendra
                    numeros aleatorios, cada numero representa un trimestre
              int maxTrim;
              int maxSeriacion;
              int maxMinimos;
              int maxExcedidos;
              int creditosTrimestre[numeroTrimestres];


              int maximoTrimestre(); //funcion que regresa un entero el cual sera el
                    numero maximo que representa el trimestre

              int SiguienteAleatorioEnteroModN(long * semilla, int n); // Funcion que
                    genera un numero aleatorio, Devuelve un entero entre 0 y n-1

              void crearTrimestre(int nTrimestre); //Funcion  que agrega a la lista
                    de trimestres Aleatorios un numero aleatorio que genera otra
                    funcion

              void funcionObjetivo(vector<UEA>listaDatos, vector<UEA>
                    listaCreditosMinimos, vector<Seriacion> listaSeriacion);//esta
                    funcion es la que va a calcular el costo de la solucion


      public:
             Solucion (vector<UEA> listaDatos, vector<UEA> CreditosMinimos, vector<
                   Seriacion> listaSeriacion); //Constructor numeroUEAs es el tamano
                   del vector

             Solucion (vector<Solucion> listaSoluciones, vector<UEA> listaDatos,
                   vector<UEA> listaCreditosMinimos, vector<Seriacion> listaSeriacion
                   ); //Constructor

             int tamanoTriAleatorio();

             int posTriAleatorio (int j);

             int getCosto();

             void llenaCreditosTrimestre (vector<UEA> listaDatos);

             int violacionesPorCreditosExcedidos(vector<UEA> listaDatos); //funcion
                   que verifica si nos pasamos de creditos en cada trimestre

             int violacionesCreditosMinimos(vector<UEA> listaDatos, vector<UEA>
                   listaCreditosMinimos);

             int ViolacionesSeriacion (vector<UEA> listaDatos, vector<Seriacion>
                   listaSeriacion);

             void imprime(); //funcion que imprime la solucion en pantalla, es decir
                   cada trimestre aleatorio

             void imprime2(vector<UEA> listaDatos);

             void imprimeCostos(int sol);

                  };


class trimPos {


       public:
              int nTrim;
              int indice;
              trimPos(int a, int b);

              int operator<(const trimPos &otroObjeto) const;

       };
```

Listado 2: Solucion.h

G.3 SERIACION.CPP

```
#include "Seriacion.h"

using namespace std;

Seriacion::Seriacion(){
           }
             int Seriacion::operator<(const Seriacion &otroObjeto) const{

               //En este caso diremos que el objeto actual es "menor que" el
               //otroObjeto si la clave de este es menor que la del otro.
               if (claves[o] < otroObjeto.claves[o])
                  return 1;
               return o;
            }
```

Listado 3: Seriacion.cpp

G.4 SERIACION.H

```
#include <vector>

using namespace std;

class Seriacion{//Creamos una clase Seriacion para agregar sus elementos a una lista
      y acceder a ellos


      public:
             vector<int> claves;
             Seriacion(); //constructor
             int operator<(const Seriacion &otroObjeto) const;
      };
```

Listado 4: Seriacion.h

G.5 UEA.CPP

```
#include "UEA.h"

             UEA::UEA(){ //Constructor
             }
             UEA::UEA(int cl, int cr){ //Constructor con parametros que recibiran a
                   los elementos de la clase UEA
                      clave=cl;
                      creditos=cr;
                      }
             int UEA::getClave(){ //Funcion que obtiene el valor de clave
                 return clave;
                 }
             int UEA::getCreditos(){ //Funcion que obtiene el valor de creditos
                 return creditos;
                  }
             int UEA::setClave(int cl){//Funcion a la cual asignamos el valor de
                   clave
                 clave=cl;
             }
             int UEA::setCreditos(int cr){//Funcion a la cual asignamos el valor de
                   creditos
                 creditos=cr;
             }
             int UEA::dame(){
             return clave;
             }
             int UEA::operator<(const UEA &otroObjeto) const{

                if (clave < otroObjeto.clave)
                  return 1;
               return o;
            }
```

Listado 5: UEA.cpp

G.6 UEA.H

```
1
2  class UEA{
3        private:
4                int clave;
5                int creditos;
6
7        public:
8               UEA(); //Constructor
9
10              UEA(int cl, int cr); //Constructor con parametros que recibiran a los
                      elementos de la clase UEA
11
12              int getClave(); //Funcion que obtiene el valor de clave
13
14
15              int getCreditos(); //Funcion que obtiene el valor de creditos
16
17              int setClave(int cl);//Funcion a la cual asignamos el valor de clave
18
19             int dame();
20              int setCreditos(int cr);//Funcion a la cual asignamos el valor de
                      creditos
21
22              int operator<(const UEA &otroObjeto) const;
23
24       };
```

Listado 6: UEA.h

G.7 PROGRAMA PRINCIPAL.CPP

```
1  #include "Solucion.h"
2  #include <iostream>
3  #include <fstream>
4  #include <string>
5  #include <vector>
6  #include <sstream>
7  #include <algorithm>
8  #include <ctime>
9  #include <time.h>
10
11 int RangoMin[100], RangoMax[100];
12 int TrimestreMaximo = 37;
13 float z, z1 = 0;
14 int resta;
15
16 int main(){
17
18   vector<UEA> listaDatos; //Se crea un vector de tipo UEA llamado listaDatos para
           guardar los elementos de una UEA
19   int clave; //Se crea la variable clave para poder manipular estos elementos en
           diferentes operaciones
20   int creditos;//Se crea la variable creditos para poder manipular estos elementos en
           diferentes operaciones
21   UEA materia; //Se crea el objeto materia de tipo UEA para guardar cada elemento en
           este claso Clave  y Creditos
22   string linea;//Se declara una variable llamada "linea" de tipo string para leer por
           linea el archivo Datos
23   ifstream miArchivo ("Datos.txt");//lectura de archivo Datos
24   if (miArchivo.is_open())
25   {
26     while ( getline (miArchivo,linea) )
27     {
28      istringstream parser(linea); //Se pasa de cadena a entero la variable "linea" y
              se lee cada pedazo del renglon del archivo
29        while(parser >> clave >> creditos){ //El primer elemento que encuentre lo
              asigna a clave y  el segundo elemento lo asigna a creditos
30
31                      materia.setClave(clave); // Se asigna la clave a el objeto
                            materia
32                      materia.setCreditos(creditos); // Se asigna los creditos  a el
                            objeto materia
33
34                      listaDatos.push_back(materia);
35
36          }
37      }
38    }
39     miArchivo.close(); //Cierre del archivo Datos
40
41
42     for(int i=0; i<100; i++){
43         RangoMax[i] = 36;
44         RangoMin[i] = 1;
45      }
46
47
48 //Archivo Creditos Minimos
49    vector<UEA> listaCreditosMinimos; //Se crea un vector de tipo UEA para guardar los
           elementos  del archivo Creditos Minimos
```

```
50    ifstream miArchivo2 ("Creditos_Minimos.txt");//Lectura del archivo creditos
          minimos
51    if (miArchivo2.is_open())
52    {
53      while ( getline (miArchivo2,linea) )
54      {
55        istringstream parser(linea);//Se pasa de cadena a entero la variable "linea" y
              se lee cada pedazo del renglon del archivo creditos minimos
56        while(parser >> clave >> creditos){
57
58                      materia.setClave(clave);
59                      materia.setCreditos(creditos);
60                      listaCreditosMinimos.push_back(materia);
61        }
62      }
63    }
64
65     miArchivo2.close();
66
67
68    //Archivo Seriacion
69    vector<Seriacion> listaSeriacion; //Se crea un vector de tipo Seriacion para
          agregar
70
71    ifstream miArchivo3 ("Seriacion.txt");
72    if (miArchivo3.is_open())
73    {
74      while ( getline (miArchivo3,linea) )
75      {
76        Seriacion renglon; //Se crea un objeto llamado renglon de tipo Seriacion para
              guardar cada elemento que encuentre en un renglon del archivo
77        istringstream parser(linea);
78        while(parser >> clave ){
79              renglon.claves.push_back(clave); //Se utiliza la funcion agrega para
                    incluir cada elemento de una linea del archivo Seriacion en el
                    objeto renglon
80         }
81        listaSeriacion.push_back(renglon); //Se agrega cada renglon a la lista
              Seriacion
82
83      }
84      miArchivo3.close(); // Cierre del archivo Seriacion
85     } else
86           cout << "No se pudo abrir el archivo";
87
88
89    sort(listaDatos.begin(), listaDatos.end());
90    sort(listaSeriacion.begin(), listaSeriacion.end());
91    sort(listaCreditosMinimos.begin(), listaCreditosMinimos.end());
92
93    Solucion mesa(listaDatos, listaCreditosMinimos, listaSeriacion); // se crea un
          objeto de tipo solucion que  recibira la lista Datos
94
95
96    int nSolucion;
97    int i;
98
99    //Inicia tiempo de ejecucion
100
101
102   clock_t start, end;
103   start = clock();
104
105   //***********************************************
106   cout << "\nCuantas soluciones desea?  " ;
107   cin  >> nSolucion;
108
109   if(nSolucion < 3){
110     cout << "\nError, debe proporcionar tres o mas soluciones" << endl;
111     cout << "\nPor favor repita, Cuantas soluciones desea?  " ;
112        cin  >> nSolucion;
113     }
114   else{
115        cout << "\nEspere un momento, generando soluciones...\n ";
116        }
117
118
119   vector<Solucion> listaSoluciones;
120
121   for(int i=0; i<nSolucion; i++){
122   Solucion Sol(listaDatos, listaCreditosMinimos, listaSeriacion);
123   listaSoluciones.push_back(Sol);
124     }
125
126
127   for(int i=0; i<listaSoluciones.size(); i++){
128
129                                       }
130
131     int indiceMin;
132
133     for (int n=0; n<80000; n++){
134         Solucion nuevaSol(listaSoluciones, listaDatos, listaCreditosMinimos,
              listaSeriacion);
135
136        //Sacar maximo para poder sustituir la peor solucion
```

```
137             int max=0;
138             int indiceMax=-1;
139             int costo;
140             int minCosto;
141             int costoNuevo;
142 
143 
144             for(int i=0; i<listaSoluciones.size(); i++){
145 
146                  costo = listaSoluciones[i].getCosto();
147                  if(costo > max){
148                     max=costo;
149                     indiceMax = i;
150                   }
151             } // fin for
152            int costoNuevaSol;
153            costoNuevaSol = nuevaSol.getCosto();
154 
155 
156 
157            if(costoNuevaSol <= max){
158                listaSoluciones[indiceMax]= nuevaSol;
159                minCosto= listaSoluciones[0].getCosto();
160                indiceMin=0;
161                for(int i=0; i<listaSoluciones.size(); i++){
162                costoNuevo =  listaSoluciones[i].getCosto();
163                if(costoNuevo < minCosto){
164                    minCosto=costoNuevo;
165                    indiceMin = i;
166 
167 
168                 } //fin if costoNuevo
169              } // fin for
170 
171            }  // fin if
172 
173 
174         if(n==79999){
175         cout << "\n\t\tMejor Solucion \n" << endl;
176 
177                  }
178                  listaSoluciones[indiceMin].imprimeCostos(n);
179 
180 
181 
182     }
183 
184         listaSoluciones[indiceMin].imprime2(listaDatos);
185 
186 
187 end = clock();
188 resta = end - start;
189 z = (float)resta / (float)CLOCKS_PER_SEC;
190 printf("\n\nEl tiempo de ejecucion fue de: %.4f segundos\n", z-z1);
191 
192     cin.get();
193     cin.get();
194 };
```

Listado 7: Programa Principal.cpp

H

ENTREGABLES

Se entregará tres discos compactos al Coordinador de Estudios de Ingeniará en Computación que incluirán el reporte final del proyecto terminal en un archivo PDF (sin restricciones)[1] y el código fuente de la aplicación en un archivo comprimido (sin restricciones)[2].

Los planes de estudio de la Universidad Autónoma Metropolitana unidad Azcapotzalco (UAM-A) están formados por UEA que deben ser cursadas por los estudiantes en un número preestablecido de trimestres. Cada UEA tiene asociado un número de créditos y en algunos casos requisitos para poder cursarla. El programa desarrollado nos arroja una recomendación para organizar de manera adecuada la carga académica a lo largo de la licenciatura.

La figura 2 nos muestra una solución factible, obtenida del programa Proyecto_Busqueda_Armonica.

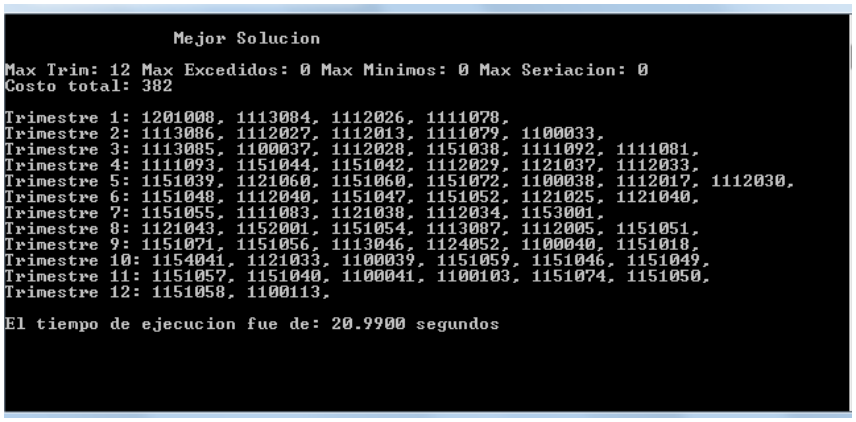


Figura 2: Resultado Final.

Está solución nos indica que UEA meter en cada trimestre, para poder concluir la licenciatura en un plazo de 12 trimestres, sin cometer ninguna violación.

1 Se debe poder visualizar, imprimir, copiar y pegar sin restricciones tecnológicas.
2 Se debe poder descomprimir sin restricciones tecnológicas.

BIBLIOGRAFÍA

[1] M. A. Asadilla, Khader A. T., M. A. Al Betar, and A. L. Bolaji. "Global best harmony search with a new pitch adjustment designed for Nurse Rostering". Journal of King Saud University Computer and Information Sciences. Vol. 25 No 2, pp. 145–162. 2013.

[2] V. Cacchiani and P. Toth. "Nominal and robust train timetabling problems". European Journal of Operational Research. Vol. 219 No 3, pp. 727-737. 2012.

[3] D. Barrera, N. Velasco and C. A. Amaya. .A network-based approach to the multi-activity combined timetabling and crew scheduling problem: Workforce scheduling for public health policy implementation". Computers & Industrial Engineering. Vol. 63 No 4, pp. 802-812. 2012.

[4] A. Salwani and T. Hamza. "On the use of multi neighbourhood structures within a Tabu-based memetic approach to university timetabling problems". Information Sciences. Vol. 191, pp. 146-168. 2012.

[5] M. Dell'Amico, J. C. Díaz Díaz, and M. Iori. "Bin packing problem with precedence constraints". Operations Research. Vol. 60 No 6, pp. 1491-1504. 2012.

[6] M. Chiarandini, L. Gaspero, S. Gualandi, and A. Schaerf. "The balanced academic curriculum problem revisited". Journal of Heuristics. Vol. 8 No 1, pp. 119-148. 2012.

[7] J. D., Castillo Cruz. .Adaptación de una técnica heurística para resolver un problema de programación de horarios". Distrito Federal : Tesis de licenciatura Universidad Autónoma Metropolitana, 2013.