Universidad Autónoma Metropolitana

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

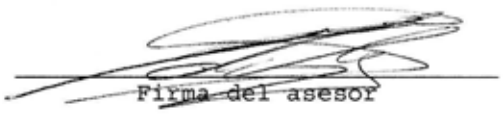
Proyecto Tecnológico

Recocido Simulado Para Resolver un Problema de Currículo Académico

Alfredo Siliceo Pérez
210330970

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

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, Alfredo Siliceo Pérez, 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.

Alfredo Siliceo P.

Firma del alumno

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Los planes de estudio de las carreras en la Universidad Autónoma Metropolitana, han sido elaborados de tal forma que el alumno avance en la acumulación de conocimientos conforme va aprobando las Unidades de Enseñanza y Aprendizaje (UEA). Sin embargo, la flexibilidad de esto planes permite que el alumno inscriba materias de distintos trimestres, sin reflexionar en la importancia del conocimiento previo que debe tener para cada UEA. Esto puede ocasionar que el alumno deje inconclusa la materia, incrementando su índice de no aprobación, y en consecuencia, repercutir en la eficiencia terminal.

En este proyecto de Integración se adaptó la técnica heurística Recocido Simulado“, para generar propuestas, que indiquen el trimestre en el cual debe cursarse cada una de las UEA seleccionada por un estudiante, de las carreras de Ingeniería en Computación e Ingeniería Física, de la Universidad Autónoma Metropolitana unidad Azcapotzalco. Para cada propuesta realizada, el algoritmo busca minimizar el número de trimestres requeridos, tomando en consideración restricciones como seriación, máximo número de créditos por trimestre, y mínimo número de créditos requeridos para cursar algunas UEA.

El objetivo de este proyecto es proporcionar una herramienta computacional, que le indique al alumno el trimestre en el cual debe cursar las UEA que ha seleccionado, así como el tiempo mínimo requerido para completar la carrera. Para lograrlo, se 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 Recocido Simulado 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.

1 INTRODUCCIÓN

El problema de asignación de horarios puede entenderse como la asignación, sujeta a restricciones, de recursos a ciertas tareas disponibles dentro de ventanas de tiempo específicas, de tal forma que se satisfacen, en la medida de lo posible, un conjunto de objetivos deseables. Este tipo de problemas aparece en diferentes actividades diarias, como generar turnos de enfermeras, horarios en medios de transporte y formación de equipos de trabajo. En particular es común en instituciones educativas de nivel superior, y normalmente implica la asignación de profesores, aulas, laboratorios, cursos, exámenes, entre otros, a horarios establecidos.

Un caso especial 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 optimice una función objetivo, como cargas académicas balanceadas durante toda la carrera.

En este trabajo se presenta un algoritmo basado en la técnica heurística Recocido Simulado para resolver un conjunto de ejemplos del problema de currículo académico, en el cual se busca ubicar unidades de enseñanza y aprendizaje (UEA) en diferentes trimestres, de tal forma que se satisfacen las condiciones, establecidas por la División de Ciencias Básicas de la Universidad Autónoma Metropolitana unidad Azcapotzalco (UAM-A), para que los estudiantes puedan inscribirse en ellas, al tiempo que se busca minimizar el número de trimestres requeridos para completarlas.

2 JUSTIFICACIÓN

Determinar el orden en el que deben tomarse los cursos para completar una licenciatura, al tiempo que se minimiza el número de trimestres requeridos, es un problema NP-Duro [1][1]. Por lo cual, el uso de métodos exactos, o manuales, puede ser inadecuado en ejemplos de la vida real, ya que encontrar una buena solución puede requerir mucho tiempo[2]. Por lo tanto, diseñar una herramienta capaz de generar soluciones de forma automática y eficiente se ha convertido en un objetivo valioso[3].

1 Existe una transformación polinomial entre este problema y el problema de empaquetamiento con restricciones de precedencia y viceversa.

2 Una buena solución implica respetar los lineamientos establecidos por UAM-A, como seriaciones y otros, al tiempo que se minimiza el número de trimestres empleados.
GUSEK, un programa de cómputo para optimización lineal de licencia libre, no encontró el óptimo de un ejemplo, que se empleará en este proyecto, después de 48 horas de ejecución.

3 Esto puede ir más allá de las capacidades del boligrama, cuando el número de UEA incluidas en el ejemplo sea mayor que el mínimo requerido para completar la licenciatura, o cuando exista una solución que requiera menos de 13 trimestres para cursar las UEA seleccionadas.

Por lo anterior, el uso de técnicas heurísticas es una opción que permite encontrar soluciones de buena calidad en tiempos de cómputo aceptables. Con la propuesta realizada en este trabajo se obtendrá 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[4]. El algoritmo diseñado se basará en Recocido Simulado, una técnica heurística cuya eficiencia se ha comprobado en diferentes aplicaciones. Los ejemplos empleados en este proyecto se generarán a partir del plan de estudios de las carreras de ingeniería en computación e ingeniería física de la UAM-A. Esto implica que las soluciones obtenidas para ejemplos que contengan el mínimo número de UEA requerido para que un estudiante complete su licenciatura, podrían coincidir con los diagramas de seriación de la carrera, en el caso de que al ejecutar el algoritmo, la carga académica se distribuya en 13 trimestres. Asimismo, se emplearán dos proyectos de integración ya concluidos, [2, 3], para comparar el desempeño o la calidad de las soluciones obtenidas en esta propuesta. Se debe destacar que en estos proyectos se adaptaron técnicas heurísticas que originalmente habían sido diseñadas para resolver problemas continuos. Sin embargo, el problema de currículo académico es discreto, por lo que una técnica de búsqueda local, como Recocido Simulado, puede ser más eficiente. Aunado a lo anterior, hasta donde se tiene conocimiento, Recocido Simulado no se ha empleado para resolver el problema estudiado en este proyecto.

3 OBJETIVOS

3.1 Objetivo General

Implementar un algoritmo inspirado en la heurística Recocido Simulado para resolver algunos ejemplos del problema de currículo académico[5].

3.2 Objetivos específicos

1. Diseñar una codificación adecuada de las variables del problema.
2. Definir adecuadamente una función de vecindario acorde a las características del problema.
3. Adaptar la técnica heurística Recocido Simulado al problema de currículo académico.

4 Factores como reprobación, traslape entre UEA en un trimestre, o UEA que no se abren pueden aumentar el tiempo real requerido para completar la licenciatura. Sin embargo, modelar estas características se encuentran más allá del alcance de la propuesta actual, por lo cual no se han incluido.

5 En este trabajo, un ejemplo del problema de currículo académico, es el conjunto de UEA que un alumno desea cursar durante toda la licenciatura.

4. Calibrar el algoritmo sobre un conjunto reducido de ejemplos.
5. Implementar un algoritmo basado en la técnica de Recocido Simulado.
6. Analizar los resultados y comparar el desempeño de la técnica propuesta con el de algoritmos actualmente operantes.

4 MARCO TEÓRICO

El recocido simulado es un algoritmo en el mismo sentido en que lo son las reglas para multiplicar o dividir dos números, pero se trata de un conjunto de operaciones mucho más complicado, de manera que en la práctica sólo puede ejecutarse mediante máquinas de cálculo.

El recocido simulado busca mejorar alguna de las propiedades de un objeto, consiste en producir al azar cambios en sus valores.

La simulación del proceso puede usarse para describir la generación de soluciones de un problema de optimización combinatoria, en donde conforme el proceso cambia (azar) y teniendo como resultado la mejor solución.

Las soluciones de un problema de optimización combinatoria son equivalentes a los estados de un sistema físico. El costo de una solución es equivalente a la energía de un estado.

El recocido simulado consiste en una búsqueda por entornos, permite aceptar soluciones peores en función de una probabilidad que va disminuyendo con el tiempo. De esta forma, al principio se realiza una búsqueda con mayor diversificación y al final se intensifica la búsqueda, al ser más difícil que se acepte una solución peor.

En cada iteración, el método evalúa algunos vecinos del estado actual denominado **S**, y probabilísticamente decide entre efectuar una transición a un nuevo estado **S'** o quedarse en el estado inicial **S**. La comparación entre estados vecinos se repite hasta que se encuentre un estado óptimo que minimice la energía del sistema o hasta que se cumpla cierto tiempo computacional u otras condiciones.

5 DESARROLLO DEL PROYECTO

5.1 Descripción de Recocido Simulado

Recocido simulado es una de las técnicas heurísticas más conocidas, que por su simplicidad y buenos resultados en numerosos problemas, se ha convertido en una herramienta muy popular, con aplicaciones en diferentes áreas de optimización. El concepto fue introducido en el campo de la optimización combinatoria a inicios de la década de los 80 por Kirkpatrick [13] y Cerny [14]. Esta heurística, se inspira en una analogía entre el proceso de recocido de sólidos y la forma en que se resuelven problemas de optimización combinatoria. Dicha analogía resulta importante para comprender la forma en que trabaja este algoritmo.

El recocido de sólidos es un proceso de tratamiento térmico que se aplica a varios materiales como el vidrio y ciertos metales y aleaciones para hacerlos menos quebradizos y más resistentes a la fractura. El objetivo de este proceso es minimizar la energía interna de la estructura atómica del material y eliminar posibles tensiones internas provocadas en las etapas anteriores de su procesado. Para lograrlo, los metales ferrosos y el vidrio se recuecen calentándolos a alta temperatura y enfriándolos lentamente. Cada vez que se baja la temperatura, las partículas se reacomodan en estados de más baja energía hasta que se obtiene un sólido con partículas acomodadas conforme a una estructura con energía mínima.

El algoritmo de recocido simulado está basado en técnicas de Monte Carlo y genera una sucesión de estados del sólido de la siguiente manera. Dado un estado actual i del sólido con energía S_i, entonces el siguiente estado j es generado al aplicar una pequeña perturbación al estado actual. La temperatura del nuevo estado es S_j. Si la diferencia de energía $S_i - S_j$, es menor o igual que cero, el estado j es aceptado como estado actual. Si la diferencia es mayor que cero, el estado j es aceptado con una probabilidad dada por:

$$\exp\left(\frac{S_i - S_j}{k_B T}\right) \tag{1}$$

Donde T denota la temperatura a la cual se encuentra el sólido y k_B es una constante física llamada la constante de Boltzmann. Este criterio de aceptación es conocido como el criterio de Metrópolis. Si el decremento de la temperatura es suficientemente lento, el sólido alcanzará un equilibrio térmico en cada temperatura.

La técnica de recocido simulado, se inspiró en el hecho de que el algoritmo de Metrópolis puede ser utilizado para generar soluciones a problemas de optimización combinatoria si se hacen las siguientes consideraciones:

- Las soluciones del problema de optimización son equivalentes a los estados del sólido.
- El costo de una solución es equivalente a la energía de cada estado.
- La temperatura será sustituida por un parámetro de control que regulará la probabilidad de aceptación de nuevas soluciones.

El algoritmo de Recocido Simulado comienza con una solución inicial y una temperatura inicial T_0. Se recomienda que al inicio del algoritmo la temperatura sea suficientemente alta para permitir todo, o casi todo, movimiento, es decir, que la probabilidad de pasar de la solución actual E_i a la solución vecina E_j sea muy alta, sin importar la diferencia entre los costos de ambas soluciones, $f(E_i) - f(E_j)$.

En cada iteración se genera una solución vecina de manera aleatoria, si la nueva solución mejora el valor de la función objetivo con respecto a la solución actual, esta última es reemplazada. Cuando la nueva solución no mejora el valor de la función objetivo, se puede

aceptar el cambio de la solución actual con cierta probabilidad dada por:

$$\exp\left(\frac{f(E_i) - f(E_j)}{T}\right) \tag{2}$$

Donde, $f(E_i)$ es el costo de la solución actual, $f(E_j)$ es el costo de la solución vecina y T es la temperatura del proceso. Conforme el algoritmo avanza, el valor de la temperatura disminuye mediante un coeficiente de enfriamiento, $0 < \alpha < 1$, pero cada valor de T se mantiene constante durante L iteraciones, para permitir que el algoritmo explore distintas soluciones con la misma probabilidad de aceptación.

Debe observarse que al inicio, cuando la temperatura es alta, se tiene una mayor probabilidad de aceptar soluciones de menor calidad, lo cual permite la exploración del espacio de soluciones y evita la convergencia prematura a mínimos locales. Sin embargo, conforme el valor de la temperatura disminuye, el algoritmo se hace más selectivo y difícilmente acepta soluciones de menor calidad, iniciando una búsqueda que lo guía hacia un mínimo local. Finalmente, el algoritmo se detiene cuando la temperatura alcanza un valor límite, T_f, y devuelve la mejor solución encontrada.

Los parámetros de temperatura inicial T_0, coeficiente de enfriamiento α, temperatura final T_f y número de soluciones visitadas en cada temperatura L, son conocidos como *programa de enfriamiento* y juegan un papel importante en el desarrollo del algoritmo. Si se utilizan valores demasiado grandes el tiempo de ejecución podría ser excesivo, mientras que valores demasiado pequeños podrían provocar una convergencia prematura a un mínimo local. Para mayores detalles sobre esta técnica ver [15].

Sea T_k denote el valor de la temperatura y L_k el número de transiciones generadas en la *k*-ésima iteración del algoritmo de Metropolis. El algoritmo de Recocido Simulado puede describirse en pseudo-código como se muestra en la Tabla 5.1.

El algoritmo de Recocido Simulado comienza llamando a un procedimiento de inicialización donde se definen la solución inicial, parámetro de control inicial y el número inicial de generaciones necesarias para alcanzar el equilibrio térmico para la temperatura inicial. La parte medular del algoritmo consta de dos ciclos. El externo **Repite**...**hasta** y el interno **Para**...**fin para**. El ciclo interno mantiene fija la temperatura hasta que se generan L_k soluciones y se acepta o se rechaza la solución generada conforme el criterio de aceptación ya discutido. El ciclo externo disminuye el valor de la temperatura mediante el procedimiento CALCULA-CONTROL y calcula el número de soluciones a generar para alcanzar equilibrio térmico mediante el procedimiento CALCULA-LONGITUD. Este ciclo finaliza cuando la condición de paro se cumple.

Un rasgo característico del algoritmo de Recocido Simulado es que, además de aceptar mejoras en el costo, también acepta soluciones de peor costo. Inicialmente, para valores grandes de T, puede aceptar grandes deterioros en el costo de la función objetivo; cuando T decrece, únicamente pequeñas desviaciones serán aceptadas y finalmente, cuando el valor de T se aproxima a cero, no se aceptarán desviaciones.

Comienza
INICIALIZA ($E_{i_{inicial}}, T_0, L_0$)
$k := 0$
$E_i := E_{i_{inicial}}$
Repite
Para $l := 1$ a L_k **haz**
Comienza
GENERA (E_j vecino de E_i)
si $f(E_j) \leqslant f(E_i)$ **entonces** $E_i := E_j$
si no
si $\exp\left(\frac{f(E_i)-f(E_j)}{T_k}\right) >$ número aleatorio en [0,1)
entonces $E_i := E_j$
fin para
$k := k + 1$
CALCULA-LONGITUD (L_k)
CALCULA-CONTROL (T_k)
hasta criterio de paro
fin

Tabla 1: Algoritmo de Recocido Simulado en pseudo-código

Este hecho significa que el algoritmo de Recocido Simulado tiene la capacidad de escapar de mínimos locales.

Note que la probabilidad de aceptar desviaciones está implementada al comparar el valor de $\exp(f(E_i) - f(E_j))/T$ con un número aleatorio generado de una distribución uniforme en el intervalo [0,1). Además, debe ser obvio que la velocidad de convergencia del algoritmo está determinada al escoger los parámetros L_k y T_k, $k = 0,1,....$ Si los valores T_k decrecen rápidamente o los valores de L_k no son grandes, se tendrá una convergencia más rápida que cuando los valores de T_k decrecen lentamente o los valores de L_k son grandes.

5.1.1 *Recocido Simulado aplicado al problema de currículo académico*

Durante todo el proceso de optimización, el algoritmo trabaja con una solución representada por un vector n-dimensional, donde n es el número de UEA en la instancia analizada. La solución inicial es generada al asignar un trimestre aleatorio, entre 1 y 36, a cada UEA. La solución inicial es evaluada por la función objetivo y su costo es guardado en memoria para futuras comparaciones.

Para generar una solución vecina, NS, a partir de la solución actual, AS, se elige de forma aleatoria una UEA, i, y se cambia de trimestre mediante la suma de un número entero generado de forma aleatoria en el intervalo [-2,2]. De esta forma, AS y NS sólo se diferencian en la asignación de una UEA.

Posteriormente, la solución vecina es evaluada mediante la función objetivo. Si el costo de la nueva solución es mejor que el costo de la solución actual, *AS* es reemplazada. En caso contrario, se recurre al criterio de Metropolis.

Este proceso se repite hasta alcanzar la temperatura final. Finalmente, se imprime la mejor solución generada por al algoritmo.

En la Figura 1 se presenta el diagrama de flujo del algoritmo desarrollado.

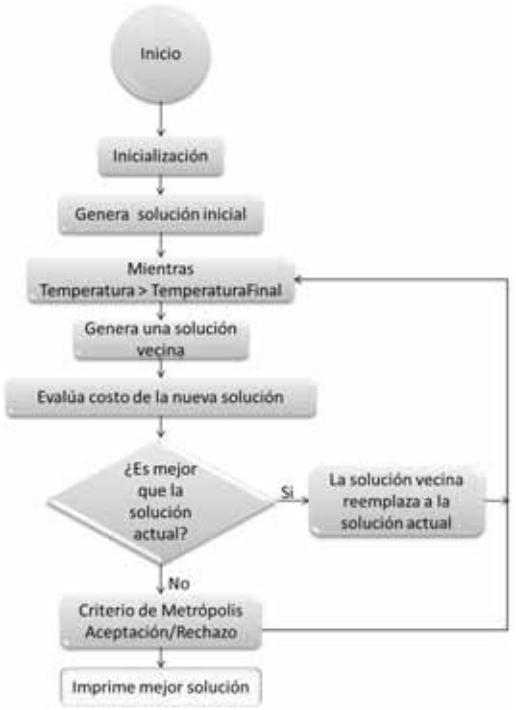


Figura 1: Diagrama de flujo Recocido Simulado.

5.2 Descripción del Problema

Los planes de estudio de la Universidad Autónoma Metropolitana están formados por UEA que deben ser cursadas por los estudiantes en un máximo de 12 años. Cada UEA tiene asociado un número de créditos y en algunos casos requisitos para poder cursarla. Estos requisitos se dividen en dos grupos:

R1) Seriación. La inscripción a algunas UEA está sujeta a la acreditación previa de otras unidades, esto con el fin de que los estudiantes cuenten con los conocimientos básicos necesarios para el correcto aprendizaje de cada UEA.

R2) 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 créditos que puede llevar un estudiante durante cada trimestre.

R3) Máximo 46 créditos en el primer trimestre.

R4) Máximo 60 créditos en los trimestres restantes.

Estos límites se han establecido para evitar que los estudiantes soliciten una carga académica excesiva, lo cual podría repercutir negativamente en su desempeño global.

De esta forma, el problema estudiado en este proyecto, consiste en encontrar el mínimo número de trimestres requeridos para cursar un conjunto de UEA, de tal forma que se respetan las restricciones **R1-R4**.

5.3 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.

5.4 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.

5.5 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.

5.6 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.

5.7 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 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.

7 PRUEBAS

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

1. Procesador: AMD A4-3300M
2. Memoria RAM: 8 GB DDR3
3. Sistema Operativo: Windows 7

7.1 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.

7.2 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 Recocido Simulado descrito anteriormente. La temperatura inicial se fijo en 5, la temperatura final en 0.0007, y el factor de enfriamiento en 0.95. Finalmente, se reporta el tiempo de ejecución y la calidad de la solución entregada.

En la sección de Anexos, se presentan 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 no se viola ninguna de las restricciones)

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

4. MaxMinimos: corresponde a una violación, el número obtenido hace referencia a cuántas 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.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

8 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 2, 3 y 4 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 36 segundos en promedio.

Se observa que la instancia más complicada fue la de "Concentracion Sistemas de Informacion" para la cual sólo en una corrida

se obtuvo una solución válida con el mínimo número de trimestres, mientras que la más sencilla fue "Instrumentación I", para la cual se obtuvieron, en 8 corridas, soluciones válidas que requerían de 10 trimestres.

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Algoritmos e Inteligencia Artificial	12	4	57.45005 seg
Mecatrónica	12	20	48.8923 seg
Sistemas de Información	11	1	41.66055 seg

Tabla 2: Ingeniería en Computación.

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Instrumentación I	10	8	35.3552 seg
Instrumentación II	11	14	36.49675 seg

Tabla 3: Instrumentación I y II.

Instancia	Trimestres Empleados	Soluciones Encontradas	Tiempo Promedio de Ejecución
Tecnología I	11	6	35.82885 seg
Tecnología II	11	5	36.5286 seg

Tabla 4: Tecnología I y II.

Para determinar la eficiencia del algoritmo propuesto se compararon los resultados obtenidos con los reportados en los proyectos de integración [2, 3]. Es importante destacar que en dichos proyectos se emplearon algoritmos poblacionales, el primero de ellos basado en Optimización por Enjambre de Partículas, y el segundo en Búsqueda Armónica. Asimismo, para garantizar una comparación justa entre las tres técnicas heurísticas, se emplearon las instancias de Instrumentación y Tecnología I y II, reportadas en ambos proyectos. De estas propuestas, sobresale el desempeño del algoritmo basado en Búsqueda Armónica, ya que que las mejores soluciones reportadas en [2] requieren de 13 trimestres para poder acomodar las UEA sin incurrir en violaciones, mientras que con Búsqueda Armónica se encontraron soluciones que solamente requiere de 12 y 11 trimestres. Sin embargo, el desempeño del algoritmo basado en Recocido Simulado es incluso mejor, ya que encuentra soluciones factibles que requiren menos trimestres que las reportadas en [3].

Por lo anterior, se concluye que el algoritmo basado en Recocido Simulado 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, o bien el tiempo de ejecución requerido, 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.

9 CONCLUSIONES

En este proyecto se implementó un algoritmo basado en la técnica heurística Recocido Simulado para resolver un problema de currículo académico, 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 los proyectos de integración [2, 3], se observó que la propuesta realizada en este proyecto superó el desempeño de Optimización por Enjambre de Partículas y Búsqueda Armónica, 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, y a disminuir el tiempo de ejecución requerido. 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.

ANEXOS

A INGENIERÍA EN COMPUTACIÓN.

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

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 5: Obligatorias de Ingeniería en Computación.

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 6: 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 7: 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 8: 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 9: Área de Concentración Sistemas de Información.

B INSTANCIAS INGENIERÍA FÍSICA

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

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 10: Obligatorias de Ingeniería Física.

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 11: 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 12: 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 13: 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 14: 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 15: Área de Concentración Tecnología. Instancia II.

C RESULTADOS PARA INGENIERÍA EN COMPUTACIÓN. PARTE 1.

Instancia: Área de Concentración de Ingeniería en Computación.
Trimestres Empleados: Número de trimestres en el que puedes terminar la licenciatura.
Factible: Se considera factible si el número máximo de trimestres empleados es 13.
MaxExcedidos: Número de trimestres en los cuales nos hemos pasado de los créditos permitidos.
MaxMinimos: Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas.
MaxSeriacion: Número de UEA que no cumplieron con la seriación.
Tiempo de Ejecución: Tiempo que tarda el algoritmo 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	64.000 seg
Mecatrónica	12	Si	0	0	0	53.665 seg
Sistemas de Información	11	No	4	0	0	40.799 seg

Tabla 16: Resultados con la semilla 83554.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	0	54.303 seg
Mecatrónica	12	Si	0	0	2	56.557 seg
Sistemas de Información	11	No	3	0	0	39.204 seg

Tabla 17: Resultados con la semilla 51025.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	Si	0	0	0	57.573 seg
Mecatrónica	12	Si	0	0	0	52.688 seg
Sistemas de Información	11	No	3	0	0	42.154 seg

Tabla 18: Resultados con la semilla 63896.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	1	0	0	57.462 seg
Mecatrónica	12	Si	0	0	0	52.664 seg
Sistemas de Información	11	No	3	0	3	40.017 seg

Tabla 19: Resultados con la semilla 34135.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	0	60.463 seg
Mecatrónica	12	Si	0	0	0	57.718 seg
Sistemas de Información	11	No	4	0	0	45.183 seg

Tabla 20: Resultados con la semilla 26575.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	2	58.814 seg
Mecatrónica	12	Si	0	0	0	51.989 seg
Sistemas de Información	11	No	2	0	0	41.162 seg

Tabla 21: Resultados con la semilla 74697.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	Si	0	0	0	57.334 seg
Mecatrónica	12	No	0	0	0	49.642 seg
Sistemas de Información	11	No	3	0	0	40.946 seg

Tabla 22: Resultados con la semilla 52563.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	1	0	0	58.246 seg
Mecatrónica	12	Si	0	0	0	56.713 seg
Sistemas de Información	11	No	4	0	0	40.856 seg

Tabla 23: Resultados con la semilla 33729.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	2	0	0	63.561 seg
Mecatrónica	12	No	0	0	0	55.747 seg
Sistemas de Información	11	No	5	0	0	40.372 seg

Tabla 24: Resultados con la semilla 83848.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	4	0	0	56.714 seg
Mecatrónica	12	Si	0	0	0	55.276 seg
Sistemas de Información	11	No	1	0	0	42.115 seg

Tabla 25: Resultados con la semilla 44431.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	1	0	0	54.447 seg
Mecatrónica	12	Si	0	0	0	50.541 seg
Sistemas de Información	11	No	4	0	0	39.268 seg

Tabla 26: Resultados con la semilla 48767.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	58.377 seg
Mecatrónica	12	Si	0	0	0	41.709 seg
Sistemas de Información	11	No	3	0	0	41.453 seg

Tabla 27: Resultados con la semilla 91416.

D RESULTADOS PARA INGENIERÍA EN COMPUTACIÓN. PARTE 2.

Instancia: Área de Concentración de Ingeniería en Computación.
Trimestres Empleados: Número de trimestres en el que puedes terminar la licenciatura.
Factible: Se considera factible si el número máximo de trimestres empleados es 13.
MaxExcedidos: Número de trimestres en los cuales nos hemos pasado de los créditos permitidos.
MaxMinimos: Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas.
MaxSeriacion: Número de UEA que no cumplieron con la seriación.
Tiempo de Ejecución: Tiempo que tarda el algoritmo 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	62.342 seg
Mecatrónica	12	Si	0	0	0	41.802 seg
Sistemas de Información	11	No	3	0	0	40.843 seg

Tabla 28: Resultados con la semilla 35462.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	56.705 seg
Mecatrónica	12	Si	0	0	0	41.867 seg
Sistemas de Información	11	No	3	0	0	49.445 seg

Tabla 29: Resultados con la semilla 49700.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	52.603 seg
Mecatrónica	12	Si	0	0	0	40.710 seg
Sistemas de Información	11	No	2	0	0	41.788 seg

Tabla 30: Resultados con la semilla 44077.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	4	0	0	56.403
Mecatrónica	12	Si	0	0	0	41.715 seg
Sistemas de Información	11	No	2	0	0	41.664 seg

Tabla 31: Resultados con la semilla 7007.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	Si	0	0	0	55.184
Mecatrónica	12	Si	0	0	0	41.720 seg
Sistemas de Información	11	No	3	0	0	41.839 seg

Tabla 32: Resultados con la semilla 60000.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	52.577
Mecatrónica	12	Si	0	0	0	54.066 seg
Sistemas de Información	11	No	2	0	0	42.643 seg

Tabla 33: Resultados con la semilla 21585.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	No	3	0	0	56.164
Mecatrónica	12	Si	0	0	0	41.699 seg
Sistemas de Información	11	No	4	0	0	42.294 seg

Tabla 34: Resultados con la semilla 94543.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Algoritmos e Inteligencia Artificial	12	Si	0	0	0	55.729
Mecatrónica	12	Si	0	0	0	39.358 seg
Sistemas de Información	11	Si	0	0	0	39.166 seg

Tabla 35: Resultados con la semilla 81553.

E RESULTADOS PARA INGENIERÍA FÍSICA. PARTE 1.

Instancia: Área de Concentración Concentración de Ingeniería Física.
Trimestres Empleados: Número de trimestres en el que puedes terminar la licenciatura.
Factible: Se considera factible si el número máximo de trimestres empleados es 13.
MaxExcedidos: Número de trimestres en los cuales nos hemos pasado de los créditos permitidos.
MaxMinimos: Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas.
MaxSeriacion: Número de UEA que no cumplieron con la seriación.
Tiempo de Ejecución: Tiempo que tarda el algoritmo en generar una solución.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	34.934 seg
Instrumentación II	12	Si	0	0	0	53.665 seg
Tecnología I	11	No	4	0	0	40.799 seg
Tecnología II	11	No	2	0	0	35.815 seg

Tabla 36: Resultados con la semilla 83554.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.465 seg
Instrumentación II	11	Si	0	0	0	37.603 seg
Tecnología I	11	No	2	0	0	35.945 seg
Tecnología II	11	No	2	0	0	35.459 seg

Tabla 37: Resultados con la semilla 51025.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	34.377 seg
Instrumentación II	11	Si	0	0	0	36.709 seg
Tecnología I	11	No	2	0	0	35.539 seg
Tecnología II	11	No	2	0	0	35.116 seg

Tabla 38: Resultados con la semilla 63896.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	34.839 seg
Instrumentación II	11	Si	0	0	0	36.223 seg
Tecnología I	11	No	2	0	0	35.324 seg
Tecnología II	11	No	2	0	0	36.738 seg

Tabla 39: Resultados con la semilla 34135.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	0	0	0	35.809 seg
Instrumentación II	11	Si	0	0	0	36.629 seg
Tecnología I	11	No	2	0	0	34.243 seg
Tecnología II	11	No	2	0	0	37.521 seg

Tabla 40: Resultados con la semilla 26575.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	33.649 seg
Instrumentación II	10	No	4	0	0	36.862 seg
Tecnología I	11	No	2	0	0	36.065 seg
Tecnología II	11	Si	0	0	0	37.613 seg

Tabla 41: Resultados con la semilla 74697.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	35.939 seg
Instrumentación II	11	Si	0	0	0	36.691 seg
Tecnología I	11	Si	0	0	0	34.604 seg
Tecnología II	11	No	4	0	0	36.96 seg

Tabla 42: Resultados con la semilla 52563.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	0	0	0	35.259 seg
Instrumentación II	11	Si	0	0	0	36.730 seg
Tecnología I	11	No	0	0	0	36.039 seg
Tecnología II	11	No	2	0	0	36.037 seg

Tabla 43: Resultados con la semilla 33729.

F RESULTADOS PARA INGENIERÍA FÍSICA. PARTE 2.

Instancia: Área de Concentración Concentración de Ingeniería Física.
Trimestres Empleados: Número de trimestres en el que puedes terminar la licenciatura.
Factible: Se considera factible si el número máximo de trimestres empleados es 13.
MaxExcedidos: Número de trimestres en los cuales nos hemos pasado de los créditos permitidos.
MaxMinimos: Número de UEA que requieren de tener un mínimo de créditos para poder cursarlas.
MaxSeriacion: Número de UEA que no cumplieron con la seriación.
Tiempo de Ejecución: Tiempo que tarda el algoritmo en generar una solución.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	35.576 seg
Instrumentación II	11	Si	0	0	0	36.853 seg
Tecnología I	11	No	0	0	0	34.803 seg
Tecnología II	11	No	2	0	0	36.116 seg

Tabla 44: Resultados con la semilla 83848.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	35.497 seg
Instrumentación II	11	Si	0	0	0	36.554 seg
Tecnología I	11	No	5	0	0	34.791 seg
Tecnología II	11	No	2	0	0	36.738 seg

Tabla 45: Resultados con la semilla 44431.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	Si	0	0	0	35.417 seg
Instrumentación II	10	No	2	0	0	36.626 seg
Tecnología I	11	Si	0	0	0	33.719 seg
Tecnología II	11	No	4	0	0	36.987 seg

Tabla 46: Resultados con la semilla 48767.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	1	0	0	35.456 seg
Instrumentación II	10	No	2	0	0	36.834 seg
Tecnología I	11	No	2	0	0	36.161 seg
Tecnología II	11	No	4	0	0	36.083 seg

Tabla 47: Resultados con la semilla 91416.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.399 seg
Instrumentación II	11	Si	0	0	0	36.52 seg
Tecnología I	11	No	4	0	0	36.441 seg
Tecnología II	11	No	2	0	0	36.488 seg

Tabla 48: Resultados con la semilla 35462.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.459 seg
Instrumentación II	11	Si	0	0	0	36.582 seg
Tecnología I	11	No	2	0	0	38.251 seg
Tecnología II	11	No	2	0	0	35.693 seg

Tabla 49: Resultados con la semilla 49700.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.515 seg
Instrumentación II	10	No	2	0	0	37.021 seg
Tecnología I	11	Si	0	0	0	36.552 seg
Tecnología II	11	No	8	0	0	37.533 seg

Tabla 50: Resultados con la semilla 44077.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.526 seg
Instrumentación II	10	No	2	0	0	36.614 seg
Tecnología I	11	No	2	0	0	36.490 seg
Tecnología II	11	No	8	0	0	35.880 seg

Tabla 51: Resultados con la semilla 7007.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	4	0	0	35.402 seg
Instrumentación II	11	Si	0	0	0	36.613 seg
Tecnología I	11	No	4	0	0	37.29 seg
Tecnología II	11	Si	0	0	0	35.413 seg

Tabla 52: Resultados con la semilla 60000.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.985 seg
Instrumentación II	11	Si	0	0	0	34.79 seg
Tecnología I	11	No	2	0	0	36.179 seg
Tecnología II	11	Si	0	0	0	36.495 seg

Tabla 53: Resultados con la semilla 21585.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	4	0	0	35.631 seg
Instrumentación II	10	No	2	0	0	35.412 seg
Tecnología I	11	No	2	0	0	35.788 seg
Tecnología II	11	Si	0	0	0	37.503 seg

Tabla 54: Resultados con la semilla 94543.

Instancia	Trimestres Empleados	Factible	Violaciones Cometidas			Tiempo de Ejecución
			MaxExcedidos	MaxMinimos	MaxSeriacion	
Instrumentación I	10	No	2	0	0	35.97 seg
Instrumentación II	11	Si	0	0	0	34.538 seg
Tecnología I	11	Si	0	0	0	36.270 seg
Tecnología II	11	Si	0	0	0	38.384 seg

Tabla 55: 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>
#include <fstream>

long Semilla = 81553;
int UEAsMaxTrim, MaxTrim;
ofstream fs("nombre.txt", ios::app);

            int Solucion::maximoTrimestre(){

               int max;
               if(trimestreAleatorio.size()==0){
                     cout << "Error, no hay elementos" << endl;
                     }
                     else{

                         max=trimestreAleatorio[0];
                         for(int i=0; i<trimestreAleatorio.size(); i++){
                             if (trimestreAleatorio[i] > max)
                                   max=trimestreAleatorio[i];
                             }
                         UEAsMaxTrim = 0;
                         for(int i=0; i<trimestreAleatorio.size(); i++){
                             if (trimestreAleatorio[i] == max)
                                   UEAsMaxTrim++;
                             }
                         }

                                   return max;
             }


             int Solucion::SiguienteAleatorioEnteroModN(long * semilla, int 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 Solucion::SiguienteAleatorioRealoy1(long * semilla)

             {
                 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;
                 zi = zi / mhi31;
                 return (zi);
             }


              void Solucion::crearTrimestre(int nTrimestre){

                   for(int i=0; i<nTrimestre; i++){
```

```
80                      trimestreAleatorio.push_back(SiguienteAleatorioEnteroModN(&
                            Semilla, numeroTrimestres-2)+1);
81              }
82      }
83
84
85      void Solucion::funcionObjetivo(vector<UEA>listaDatos, vector<UEA>
            listaCreditosMinimos, vector<Seriacion> listaSeriacion){
86          maxTrim = maximoTrimestre();
87          maxExcedidos= violacionesPorCreditosExcedidos(listaDatos);
88          maxMinimos = violacionesCreditosMinimos(listaDatos,
                listaCreditosMinimos);
89          maxSeriacion = ViolacionesSeriacion(listaDatos, listaSeriacion);
90          costo = 1 * maxTrim + 1 * maxExcedidos  + 1 * maxMinimos + 4 *
                maxSeriacion + (30 * maxTrim + 5 * UEAsMaxTrim);
91          costoFactible = maxExcedidos  + maxMinimos + maxSeriacion;
92
93
94      }
95
96
97      Solucion::Solucion(vector<UEA> listaDatos, vector<UEA>
            listaCreditosMinimos, vector<Seriacion> listaSeriacion){
98              crearTrimestre(listaDatos.size());
99              for(int i=0; i<numeroTrimestres; i++){
100             creditosTrimestre[i]=0;
101                }
102              llenaCreditosTrimestre(listaDatos);
103              funcionObjetivo(listaDatos, listaCreditosMinimos,
                     listaSeriacion);
104
105     }
106
107     Solucion::Solucion(vector<Solucion> listaSoluciones, vector<UEA>
            listaDatos, vector<UEA> listaCreditosMinimos, vector<Seriacion>
            listaSeriacion){
108
109       int UEAElegida;
110       int TrimestreCambio;
111       float b;
112
113       UEAElegida = SiguienteAleatorioEnteroModN(& Semilla,
              listaSoluciones[0].tamanoTriAleatorio() - 1);
114       TrimestreCambio = SiguienteAleatorioEnteroModN(& Semilla, 2) + 1;
115
116       for(int j=0; j<listaSoluciones[0].tamanoTriAleatorio(); j++){
117
118           if(j != UEAElegida)
119               trimestreAleatorio.push_back(listaSoluciones[0].
                      posTriAleatorio(j));
120           else{
121               b = SiguienteAleatorioRealoy1(& Semilla);
122               if(b < 0.5){
123                       if(listaSoluciones[0].posTriAleatorio(j) +
                              TrimestreCambio > RangoMax[UEAElegida] + 2)
124                           trimestreAleatorio.push_back(RangoMax[
                                  UEAElegida] + 2);
125                       else
126                           trimestreAleatorio.push_back(listaSoluciones
                                  [0].posTriAleatorio(j) +
                                  TrimestreCambio);
127               }
128               if(b > 0.5){
129                       if(listaSoluciones[0].posTriAleatorio(UEAElegida)
                              - TrimestreCambio < RangoMin[UEAElegida])
130                           trimestreAleatorio.push_back(RangoMin[
                                  UEAElegida]);
131                       else
132                           trimestreAleatorio.push_back(listaSoluciones
                                  [0].posTriAleatorio(j) -
                                  TrimestreCambio);
133               }
134           }
135       }
136
137     for(int i=0; i<numeroTrimestres; i++){
138         creditosTrimestre[i]=0;
139     }
140       llenaCreditosTrimestre(listaDatos);
141       funcionObjetivo(listaDatos, listaCreditosMinimos, listaSeriacion)
              ;
142     }
143
144
145     int Solucion::tamanoTriAleatorio(){
146         return trimestreAleatorio.size();
147     }
148
149
150     int Solucion::posTriAleatorio(int j){
151         return trimestreAleatorio[j];
152     }
153
154
155     int Solucion::getCosto(){
156       return costo;
```

```
157             }
158
159             int Solucion::getCostoFactible(){
160                   return costoFactible;
161             }
162
163             void Solucion::llenaCreditosTrimestre (vector<UEA> listaDatos) {
164                  int trimestre;
165                  for(int i=0; i<trimestreAleatorio.size(); i++){
166                           trimestre=trimestreAleatorio[i];
167                           creditosTrimestre[trimestre] += listaDatos[i].getCreditos
                                 ();
168                  }
169
170             }
171
172             int Solucion::violacionesPorCreditosExcedidos(vector<UEA> listaDatos){
173
174                   int max=0;
175
176                   for(int i=1; i<numeroTrimestres; i++){
177
178                           if(creditosTrimestre[i]>maxCreditos){
179                               max+=creditosTrimestre[i]-maxCreditos;
180                           }
181                   }
182                     return max;
183             }
184
185
186             int Solucion::violacionesCreditosMinimos(vector<UEA> listaDatos, vector<
                  UEA> listaCreditosMinimos ){
187                 int creditosAcumulados[numeroTrimestres];
188                 int max=0;
189                 int claveActual;
190                 int j=0;
191                 int creditosNecesarios;
192                 int credTrimAnterior;
193                 int trimAnterior;
194
195
196                 creditosAcumulados[0]= 0;
197                 for(int i=1; i<numeroTrimestres; i++){
198                 creditosAcumulados[i] = creditosTrimestre[i]+creditosAcumulados[i
                        -1];
199                   }
200
201                 for(int i=0; i<listaCreditosMinimos.size(); i++){
202                        claveActual=listaCreditosMinimos[i].getClave();
203                        while(j<listaDatos.size() && listaDatos[j].getClave() !=
                              claveActual){
204                            j++;
205                        }
206
207                     trimAnterior=trimestreAleatorio[j]-1;
208                     creditosNecesarios=listaCreditosMinimos[i].getCreditos();
209                     credTrimAnterior=creditosAcumulados[trimAnterior];
210
211                     if( (creditosNecesarios-credTrimAnterior) > 0){
212                        }
213
214                     if( (creditosNecesarios-credTrimAnterior) > 0){
215                         max+=creditosNecesarios-credTrimAnterior;
216                        }
217                 }
218                  return (max);
219             }
220
221
222         int Solucion::ViolacionesSeriacion(vector<UEA> listaDatos, vector<Seriacion>
                listaSeriacion ){
223
224             int max=0;
225             int claveMax=0;
226             int claveSeriacion;
227             int j=0;
228             int malas;
229             vector<UEA>::iterator posRelativa;
230             int indice;
231             int trimAnterior;
232             int trimActual;
233
234
235             for(int i=0; i<listaSeriacion.size(); i++){
236                    claveSeriacion=listaSeriacion[i].claves[0];
237                    while(j<listaDatos.size() && listaDatos[j].getClave() !=
                          claveSeriacion){
238                         j++;
239                    }
240
241                    malas=0;
242
243                    for(int k=1; k<listaSeriacion[i].claves.size(); k++){
244
245                    UEA elemBuscar;
246                    elemBuscar.setClave(listaSeriacion[i].claves[k]);
```

```
247
248
249                 posRelativa = lower_bound(listaDatos.begin(), listaDatos.end(),
                          elemBuscar);
250
251
252                 indice = posRelativa - listaDatos.begin();
253
254                 trimAnterior=trimestreAleatorio[indice];
255                 trimActual=trimestreAleatorio[j];
256
257                 if(RangoMax[indice] > TrimestreMaximo)
258                       RangoMax[indice] = TrimestreMaximo;
259                 if(RangoMax[j] > TrimestreMaximo)
260                       RangoMax[j] = TrimestreMaximo;
261
262                 if(RangoMax[indice] >= RangoMax[j])
263                     RangoMax[indice] = RangoMax[j] - 1;
264                 if(RangoMin[j] <= RangoMin[indice])
265                     RangoMin[j] = RangoMin[indice] + 1;
266
267                 if(trimAnterior >= trimActual ){
268                           malas += trimAnterior - trimActual + 1;
269                   }
270               }
271                           max += malas;
272                           claveMax= listaSeriacion[i].claves[o];
273            }
274
275           return (max);
276         }
277
278
279          void Solucion::imprimeCostos(int sol){
280                         cout << "Max Trim: " << maxTrim << " Max Excedidos: " <<
                                    maxExcedidos << " Max Minimos: " << maxMinimos << "
                                    Max Seriacion: " << maxSeriacion <<  endl;
281                         cout << "Costo total: " << costo << endl;
282                         if(TrimestreMaximo > maxTrim)
283                               TrimestreMaximo = maxTrim;
284
285                          fs << maxTrim << " " << maxExcedidos << " " << maxMinimos
                                    << " " << maxSeriacion << " " << costo << " ";
286                          fs.close();
287          }
288
289
290         trimPos::trimPos(int a, int b){
291                     nTrim=a;
292                     indice=b;
293         }
294
295
296        int trimPos::operator<(const trimPos &otroObjeto) const{
297
298              if (nTrim < otroObjeto.nTrim)
299                 return 1;
300              return 0;
301         }
302
303
304         void Solucion::imprime(vector<UEA> listaDatos){
305
306                         int a;
307                         int b;
308                         int clave;
309                         int c;
310
311                         vector<trimPos> trimestrePos;
312                         for(int i=0; i<trimestreAleatorio.size(); i++){
313                             a= trimestreAleatorio[i];
314                             b= i;
315                             trimPos Objeto(a, b);
316                             trimestrePos.push_back(Objeto);
317                         }
318
319                         sort(trimestrePos.begin(), trimestrePos.end());
320
321
322                         int aux = 1;
323
324                         for (int i=0; i<trimestrePos.size(); i++) {
325                              if (trimestrePos[i].nTrim == aux){
326                                   cout << "\nTrimestre " << trimestrePos[i].nTrim
                                              << ": ";
327                                   aux++;
328                                   }
329                              clave = listaDatos[trimestrePos[i].indice].getClave()
                                        ;
330                              cout << clave << ", ";
331                         }
332        }
```

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 o 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[0] < otroObjeto.claves[0])
                 return 1;
              return 0;
           }
```

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 0;
             }
```

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 #include <math.h>
11
12 double SiguienteAleatorioRealoy1(long * semilla);
13
14 int RangoMin[100], RangoMax[100];
15 int TrimestreMaximo = 37;
16 float z, z1 = 0;
17 int resta;
18 double Temperatura = 5, FactorEnfriamiento = 0.97;
19 int NumeroIteraciones = 700;
20 float u1, u2;
21 long Semilla1 = 81553;
22 float totalTiempo;
23
24
25 int main(){
26
27
28
29   vector<UEA> listaDatos;
30   int clave;
31   int creditos;
32   UEA materia;
33   string linea;
34   ifstream miArchivo ("Datos.txt");
35   if (miArchivo.is_open())
36   {
37     while ( getline (miArchivo,linea) )
38     {
39      istringstream parser(linea);
40        while(parser >> clave >> creditos){
41                      materia.setClave(clave);
42                      materia.setCreditos(creditos);
43
44                      listaDatos.push_back(materia);
45
46          }
47       }
48    }
49     miArchivo.close();
50
51
52     for(int i=0; i<100; i++){
53         RangoMax[i] = 36;
54         RangoMin[i] = 1;
55      }
56
57
58
```

```
59
60    vector<UEA> listaCreditosMinimos;
61    ifstream miArchivo2 ("Creditos_Minimos.txt");
62    if (miArchivo2.is_open())
63    {
64      while ( getline (miArchivo2,linea) )
65      {
66        istringstream parser(linea);
67        while(parser >> clave >> creditos){
68
69                    materia.setClave(clave);
70                    materia.setCreditos(creditos);
71                    listaCreditosMinimos.push_back(materia);
72        }
73      }
74    }
75
76     miArchivo2.close();
77
78
79
80    vector<Seriacion> listaSeriacion;
81    ifstream miArchivo3 ("Seriacion.txt");
82    if (miArchivo3.is_open())
83    {
84      while ( getline (miArchivo3,linea) )
85      {
86        Seriacion renglon;
87        istringstream parser(linea);
88        while(parser >> clave ){
89              renglon.claves.push_back(clave);
90         }
91        listaSeriacion.push_back(renglon);
92
93      }
94      miArchivo3.close();
95     } else
96           cout << "No se pudo abrir el archivo";
97
98
99    sort(listaDatos.begin(), listaDatos.end());
100   sort(listaSeriacion.begin(), listaSeriacion.end());
101   sort(listaCreditosMinimos.begin(), listaCreditosMinimos.end());
102
103   Solucion mesa(listaDatos, listaCreditosMinimos, listaSeriacion);
104
105
106   int nSolucion;
107   int i;
108
109
110
111   clock_t start, end;
112   start = clock();
113
114
115 cout << "\nEspere un momento, generando solucion mediante Recocido Simulado...\n ";
116
117 nSolucion = 3;
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     int indiceMin;
127     int minCosto;
128     int costoNuevaSol;
129     int CostoFactible;
130
131     minCosto = listaSoluciones[0].getCosto();
132     listaSoluciones[1] = listaSoluciones[0];
133     listaSoluciones[2] = listaSoluciones[0];
134     CostoFactible = listaSoluciones[0].getCosto();
135
136
137
138 while(Temperatura > 0.007){
139         for (int n=0; n<NumeroIteraciones; n++){
140             Solucion nuevaSol(listaSoluciones, listaDatos, listaCreditosMinimos,
                      listaSeriacion);
141
142               int max=0;
143               int costo;
144               int costoNuevo;
145
146             max = listaSoluciones[0].getCosto();
147              costoNuevaSol = nuevaSol.getCosto();
148
149             if(costoNuevaSol < max){
150                  listaSoluciones[0]= nuevaSol;
151                 }
152             else{
153                 u1 = SiguienteAleatorioRealoy1(&Semilla1);
```

```
154                 u2 = exp((max-costoNuevaSol)/Temperatura);
155                 if(u1 < u2){
156                     listaSoluciones[0]= nuevaSol;
157                 }
158               }
159
160             if(minCosto > costoNuevaSol) {
161                         listaSoluciones[1]= nuevaSol;
162                         minCosto = nuevaSol.getCosto();
163                 }
164
165             if(CostoFactible >= costoNuevaSol){
166                 costoNuevaSol = nuevaSol.getCostoFactible();
167                 if(costoNuevaSol == 0){
168                     listaSoluciones[2]= nuevaSol;
169                     CostoFactible = nuevaSol.getCosto();
170                 }
171             }
172         }
173         Temperatura = Temperatura * FactorEnfriamiento;
174
175 }
176
177
178 cout << "\n\t\tMejor Solucion Factible\n" << endl;
179 listaSoluciones[2].imprimeCostos(2);
180 listaSoluciones[2].imprime(listaDatos);
181
182
183 cout << "\n\n\n\t\tMejor Solucion \n" << endl;
184 listaSoluciones[1].imprimeCostos(1);
185 listaSoluciones[1].imprime(listaDatos);
186
187 end = clock();
188 resta = end - start;
189 z = (float)resta / (float)CLOCKS_PER_SEC;
190 totalTiempo = z - z1;
191 printf("\n\nEl tiempo de ejecucion fue de: %.4f segundos\n", totalTiempo);
192     ofstream fs("nombre.txt", ios::app);
193     fs << totalTiempo << endl;
194     fs.close();
195
196     cin.get();
197     cin.get();
198 }
199
200
201 double SiguienteAleatorioRealoy1(long * semilla)
202
203             {
204                 long double zi, mhi31 = 2147483648u, ahi31 = 314159269u, chi31 =
                        453806245u;
205                 long int dhi31;
206                 zi = *semilla;
207                 zi = (ahi31 * zi) + chi31;
208                 if(zi > mhi31)
209                 {
210                     dhi31 = (long int) (zi / mhi31);
211                     zi = zi - (dhi31 * mhi31);
212                 }
213                 *semilla = (int) zi;
214                 zi = zi / mhi31;
215                 return (zi);
216             }
```

Listado 7: Programa Principal.cpp

BIBLIOGRAFÍA

[1] M. Chiarandini, L. Di Gaspero, S. Gualandi, A. Schaerf, "The balanced academic curriculum problem revisited", *Journal Heuristics*, vol. 18, no. 1, pp. 119-148, Enero, 2011.

[2] J. D. Castillo Cruz, "Adaptación de una técnica heurística para resolver un problema de programación de horarios", División de CBI, UAM-A, D.F, 2013.

[3] T. Garduño Villaseñor, "Búsqueda armónica para resolver un problema de asignación de unidades de enseñanza y aprendizaje", División de CBI, UAM-A, D.F., 2014.

[4] E. K. Burke and S. Petrovic. Recent research directions in automated timetabling. *European Journal of Operational Research*. Vol. 140, pp. 266–280. 2002.

[5] M. A. Awadallah, Khader A. T., M. A. Al-Betar, and A. L. Bolaji. lobal best harmony search with a new pitch adjustment designed for Nurse Rosterin. *Journal of King Saud University - Computer and Information Sciences*. Vol. 25(2), pp. 145–162. 2013.

[6] 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(4), pp. 802-812. 2012.

[7] V. Cacchiani and P. Toth. Nominal and robust train timetabling problems. *European Journal of Operational Research*. Vol. 219(3), pp. 727-737. 2012.

[8] O. J. Ibarra-Rojas and Y. A. Rios-Solis. Synchronization of bus timetabling. *Transportation Research Part B: Methodological*. Vol. 46(5), pp. 599-614. 2012.

[9] N. R. Sabar, M. Ayob, G. Kendall and R. Qu. A honey-bee mating optimization algorithm for educational timetabling problems. *European Journal of Operational Research*. Vol. 216 (3), pp. 533-543. 2012.

[10] 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.

[11] M. Dell'Amico, J. C. Díaz Díaz, and M. Iori. Bin packing problem with precedence constraints. *Operations Research*. Vol. 60 (6), pp. 1491-1504. 2012.

[12] M. Chiarandini, L. Gaspero, S. Gualandi, and A. Schaerf. The balanced academic curriculum problem revisited. *Journal of Heuristics*. Vol. 8 (1), pp. 119-148. 2012.

[13] S. Kirkpatrick, C. D. Gellat and M. P. Vecchi. Optimization by simulated annealing. *Science*. Vol. 220, pp. 671-680, 1983.

[14] V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*. Vol. 45, pp. 41-55, 1985.

[15] S. G. de los Cobos, J. Goddard, M. A. Gutiérrez y A. E. Martínez. Búsqueda y exploración estocástica. Ed. México: UAM-I, 2010.