Inteligencia Artificial

Informe Final: Examination Timetabling Problem

José Pablo Southerland Silva

24 de junio de 2024

Evaluación

Código Fuente (20 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

En las instituciones de educación superior existe un problema llamado Examination Timetabling Problem, donde un conjunto de exámenes debe ser planificado en bloques horarios tal que ningún estudiante tenga más de un examen a la vez. Cada institución impondrá una lista de restricciones "blandas", que para su contexto permiten evaluar de forma positiva el modelo. Este trabajo explica y modela el problema de acuerdo a los principales avances conocidos en la literatura, junto con presentar un poco de historia y variantes del mismo.

1. Introducción

Los problemas de planificación horaria se encuentran presentes en diversos planos organizacionales e institucionales. Ejemplo de esto es en las instituciones de educación superior, donde se necesita planificar correctamente los horarios de los exámenes de los y las estudiantes. A partir de ello nace el *Examination Timetabling Problem* (en adelante, ETP), que busca evitar la superposición de exámenes de cursos con estudiantes en común, intentar aumentar la distancia entre evaluaciones lo más posible [14], entre otras restricciones. Lo interesante del problema es entender el comportamiento del mismo en base a las restricciones que se vayan agregando, y ver qué tan efectiva es la solución para dar cumplimiento a la mayor parte posible de ellas.

En este trabajo se hace una definición del problema a estudiar, describiendo sus variables y restricciones; se describen los principales avances históricos relacionados a la resolución del problema y se define el modelo matemático del mismo. Finalmente se genera una conclusión a modo de análisis y resumen de lo anteriormente descrito.

2. Definición del Problema

El problema a estudiar busca asignar exámenes a bloques de horarios que satisfagan un set de restricciones. Estas restricciones se pueden separar en dos: restricciones de primer orden o duras, y restricciones de segundo orden o blandas. Las restricciones duras corresponden a aquellas que sí o sí deben cumplirse, por ejemplo, en el caso de que un o una estudiante tenga que rendir dos exámenes programados en el mismo bloque de horario. Las restricciones de segundo orden, en tanto, corresponden a aquellas que no necesariamente deben cumplirse pero sí se desea que se cumplan. Ejemplo de esto último es cuando se desea que un estudiante tenga exámenes dentro de dos bloques no tan cercanos el uno del otro [5].

En la literatura existe una gran variedad de problemas de planificación horaria, incluyendo los ámbitos de la educación, la enfermería, los deportes y el transporte. La definición general del problema cubre la mayoría de estos casos: *"Un problema de planificación horaria es un problema con cuatro parámetros: T, un conjunto finito de tiempos; R, un conjunto finito de recursos; M, un conjunto finito de reuniones; y C, un conjunto finito de restricciones. El problema es asignar tiempos y recursos a las reuniones para satisfacer las restricciones en la medida de lo posible"* [13].

El ETP corresponde a una variante más en el ámbito de la educación, donde se puede encontrar además al *University Course Timetabling Problem* (en adelante, UCTTP), que designa la planificación horaria de las salas de clases en un entorno universitario; el *School Timetabling Problem* (también conocido como *Class-Teacher Problem*), que trata la planificación horaria de profesores y profesoras en las escuelas, con el fin de que no tengan más de una clase por bloque horario [14]. Si bien todos estos problemas pertenecen al mismo contexto, se puede hacer diferenciaciones entre ellos. Entre el UCTTP y el ETP, por ejemplo, principalmente por la adición o eliminación de ciertas restricciones en particular, como las relacionadas a la distancia entre evaluaciones (*best spread of events*), que tienen un significado distinto en un UCTTP dada la naturaleza de los problemas y las necesidades de los y las estudiantes [7].

La extensa variedad de restricciones presentes en la literatura no permite hacer un set completamente definido de las mismas, sin embargo, es posible definir las más relevantes. En el caso de las restricciones duras, no puede haber exámenes con estudiantes en común asignados simultáneamente, y los recursos deben ser suficientes (por ejemplo, el número de estudiantes en una sala debe ser menor o igual a la capacidad de la sala). Por el lado de las restricciones blandas, se desea separar lo más posible los exámenes que hacen mayor conflicto, o al menos no en bloques consecutivos o días; grupos de exámenes que se deben rendir a la misma hora, el mismo día y en una ubicación; exámenes consecutivos; programación de exámenes de larga duración, número limitado de estudiantes por bloque horario; requisitos de recursos, etc. [2, 13]

El ETP es un problema NP-completo, lo que indica que no es posible hacer una búsqueda exhaustiva en un tiempo razonable [6], pero se han desarrollado diversas técnicas heurísticas y algoritmos aproximados para poder abordarlo en la práctica [13]. La principal forma de poder abordar el problema es transformarlo a un problema de coloreo de grafos, y a partir de ello aplicar técnicas, como se verá más adelante.

3. Estado del Arte

3.1. Historia del problema

El *Timetabling Problem* general comienza a ser introducido por Schmidt & Ströhlein en 1979, reuniendo las principales investigaciones al respecto y dando las primeras luces de lo que es el problema, describiendo técnicas basadas en la simulación "humana" de resolver el problema, técnicas que también se les puede llamar heurísticas directas, que se basan en la argumentación sucesiva, buscando por ejemplo, asignar primero aquellas reuniones o clases más restringidas primero [15].

Las investigaciones directas al ETP se generan en 1986, cuando Michael Carter publicó *"A Survey of Practical Applications of Examination Timetabling Algorithms"*, basándose en los problemas existentes en las escuelas norteamericanas los últimos 20 años, donde existía un sistema flexible de cursos electivos y programas basados en las necesidades individuales de los y las estudiantes, que si bien permitía la planificación manual de dichos cursos, llegó un momento en que el crecimiento de las escuelas y la cantidad de estudiantes hacían imposible realizar este trabajo de forma manual. En la publicación hace un alcance a los principales algoritmos desarrollados, con la finalidad de reorganizar el estado del arte del problema. En lo que describe, hace hincapié en el coloreo de grafos y vértices como ayuda para abordar el problema [4]. Es la primera investigación dedicada exclusivamente al ETP, ya que si bien antes se hicieron estudios con respecto al *Timetabling Problem*, ninguno entregó información directa sobre esta variante.

El 2007 se realiza la segunda edición del *International Timetabling Competition* (ITC2007), donde los participantes desarrollaron y evaluaron algoritmos para resolver problemas de *Timetabling* en diversas áreas, como la educación, la industria y los servicios. La ITC2007 ayudó a impulsar el avance en el campo de la optimización de horarios y a establecer estándares de comparación para diferentes enfoques y algoritmos [7].

A lo largo de los años se ha dado diversos enfoques a la resolución del problema, pero los que resaltan y que tienen mejores resultados son los que se mencionan a continuación.

3.2. Métodos y técnicas estudiadas

3.2.1. Simulated Annealing

El *Simulated Annealing* (en adelante, SA) es una meta-heurística secuencial que ha sido utilizada exitosamente en la solución de problemas de coloreo de grafos. En ese contexto y tal como se decribió en la sección anterior, el coloreo de grafos es un acercamiento válido para la resolución de ETPs. El SA busca un conjunto de soluciones a través de una estructura de vecindad predefinida. A partir de una solución generada de forma aleatoria, se muestrea una solución vecina y se compara con la actual según una función de costos asociada. Si dicha función es mejorada, se acepta la nueva solución. El SA utiliza una función de probabilidad para buscar dentro de algunas soluciones inferiores (la idea es que no pase por todas). Esta función de probabilidad depende de un parámetro t, referido a la temperatura. Este valor se "enfría" o disminuye por cada ejecución. Si se permite que t sea muy pequeña, la búsqueda podría quedar atrapada en un mínimo local, por lo que una buena solución en este caso depende de la calidad de la velocidad de enfriamiento [16].

3.2.2. Fast Simulated Annealing

Siguiendo la línea del SA, el *Fast Simulated Annealing* (en adelante, FastSA) es una técnica más reciente, desarrolada en una publicación de Nuno Leite en 2019, y permite mejorar los resultados del SA. En el FastSA, cada examen seleccionado para planificación se mueve solamente (y dicho movimiento también es evaluado) si ese examen tuvo algún movimiento aceptado en el intervalo de temperatura (*temperature bin*) inmediatamente anterior.

En comparación con el SA, FastSA hace en promedio un 17 % menos de evaluaciones y un máximo de 41 % en una instancia. En cuanto a costo de solución, el FastSA logra una mejor *fitness function* en cuatro de doce instancias y requiere menos tiempo de ejecución. En comparación con otros modelos del estado del arte, mejora en una de doce instancias, y queda en el tercer lugar de los mejores 5 algoritmos [10].

3.2.3. Tabu Search

Tabu Search (en adelante, TS) es una técnica que explora un subconjunto V del vecindario $N(s_i)$. Entre los elementos de V, el que da el valor mínimo de la función de costo se vuelve

el nuevo estado s_{i+1}, independientemente de que $f(s_{i+1})$ sea menor o mayor que $f(s_i)$. Dicha elección permite al algoritmo escapar de un mínimo local, pero arriesga el entrar en un ciclo de un conjunto de estados. Para evitar esto, se utiliza una *Tabu List* o Lista Tabú, que contiene soluciones prohibidas, lo cual ayuda a hacer que el algoritmo no se atasque en un solo vecindario.

Uno de los principales problemas de esta técnica es que es muy sensible al input inicial, dado que si está lejos del óptimo global, TS puede requerir más tiempo para converger o puede quedarse atrapado en óptimos locales [8]. En lo estudiado por Azimi [1], es el algoritmo clásico con mayor efectividad de reducir el costo de encontrar soluciones, pero no el mejor para encontrar las mejores soluciones.

3.2.4. Genetic Algorithm

Los *Genetic Algorithms* (en adelante, GA) son otro tipo de técnica útil para poder resolver el problema. Este algoritmo se basa en la teoría de la evolución de Darwin; comienza con una población inicial de soluciones candidatas al problema. Esta población inicial posteriormente pasa por un proceso de refinamiento, conocido como generación. En cada generación se evalúa mediante una *fitness function* la efectividad o calidad de la solución. Posteriormente se sigue iterando en las siguientes poblaciones con soluciones candidatas hasta un criterio de fin [12].

3.2.5. Ant Colony System

El *Ant Colony System* (en adelante, ACS) se basa en el comportamiento de las hormigas al momento de generar sus rutas sucesivamente mediante feromonas. En ACS, dichas feromonas actúan como un modelo probabilístico para la construcción de soluciones. La evaporación de feromonas contrarresta la convergencia temprana al permitir la exploración del espacio de búsqueda por más tiempo y permite una adaptación dinámica a medida que el algoritmo progresa [11].

De acuerdo a lo enunciado en [1], ACS es el mejor de los métodos clásicos mostrados anteriormente para poder encontrar una mejor solución inicial con un 43.3 % de probabilidades, versus un 26.6 % de GA, 26.6 % de TS y 3.3 % de SA.

3.2.6. Sequential ACS-TS y otros métodos híbridos de ACS y TS

El *Sequential ACS-TS* (SAT) es un método híbrido presentado por Azimi [1] que mezcla ACS y TS, aprovechando las ventajas que cada técnica presenta: ACS es conocido por su capacidad de generar soluciones iniciales de alta calidad, mientras que TS se destaca por su capacidad de reducir el costo de las soluciones de manera significativa. El uso de ambas técnicas permite capitalizar estas ventajas, usando primeramente el ACS para generar una solución inicial de alta calidad y luego emplear dicha solución como punto de partida óptimo en TS.

El empleo de este método logra llegar a un 80 % de probabilidad de encontrar la mejor solución, superando por lejos a los algoritmos clásicos. En la publicación también define el *Sequential TS-ACS* y el *Hybrid ACS/TS*, que de igual manera son derivados de TS y ACS, pero que no tienen un rendimiento tan bueno como SAT, obteniendo resultados más parecidos a los algoritmos clásicos.

3.2.7. Column Generation

El método *Column Generation* (en adelante, CG) para ETP fue propuesto por Woumans el 2016 [17], y se basa en representar las soluciones factibles como una combinación lineal de columnas, donde cada columna representa una posible asignación de un conjunto de exámenes. Para ello, propone dos modelos derivados de CG: en el primero, una columna se define como una planificación de exámenes para un grupo de estudiantes único, y dos *Pricing Problems* se desarrollan para poder generar dichas columnas; posteriormente, el *Master Program* selecciona

una planificación para el grupo de estudiantes único correspondiente. El segundo modelo derivado sigue una línea algo parecida, pero trabaja con *máscaras*. Estas máscaras son producidas por un *Pricing Problem* y se generan para un grupo de estudiantes único, y posteriormente se utiliza el *Master Program* para asignar los exámenes en los horarios disponibles en dichas máscaras. El resultado de ambos modelos mejora el *spreading* para los estudiantes, pero genera versiones extra de exámenes, dado que se trabaja en grupos de estudiantes únicos. También se menciona que para simulaciones demasiado grandes, el *Master Program* es obstaculizado para lograr su resolución, y se obstaculiza el método actual de planificar exámenes para cumplir con las limitaciones de integralidad, dado que se realizan más versiones de un mismo examen.

Existen otras técnicas para poder buscar soluciones al ETP, como lo son *Memetic Algorithms*, *Particle Swarm Optimization*, *Great Deluge*, entre otros [17].

4. Modelo Matemático

El Modelo Matemático que se presenta a continuación se basa en lo propuesto en la ITC2007 [7], con algunas modificaciones para simplificar el problema, por ejemplo, permitir que la cantidad de salas sea infinita y que la duración de los exámenes sea la misma para todos, y quitando el término de exámenes largos (*largest exams*) de lo definido en la publicación.

4.1. Variables

4.1.1. Variables de decisión primarias

- Variable examen vs. bloques horarios:

$$X_{ip}^{P} = \begin{cases} 1 & \text{si el examen } i \text{ se realiza en el bloque horario } p \\ 0 & \text{en otro caso} \end{cases} \tag{1}$$

- Variable examen vs. sala:

$$X_{ir}^{R} = \begin{cases} 1 & \text{si el examen } i \text{ se realiza en la sala } r \\ 0 & \text{en otro caso} \end{cases} \tag{2}$$

4.1.2. Variables de decisión secundaria

De acuerdo a lo mencionado en la publicación, las variables de decisión secundaria son aquellas cuyos valores serán directamente forzados dada cualquier asignación a variables primarias. Se utilizarán para poder escribir las restricciones y calcular la función objetivo. Originalmente el autor definía las siguientes:

- C_s^{2R} : Penalidad para el estudiante s por tener dos exámenes en un mismo bloque horario.
- C_s^{2D} : Penalidad para el estudiante s por tener dos exámenes en un mismo día.
- C_s^{PS} : Penalidad para el estudiante s por *period spread*.

Pero en el caso de este trabajo, las penalidades vendrán dadas según la distancia que haya entre los exámenes de un mismo estudiante, vale decir:

- $C_s^{w_i}$: Penalidad para el estudiante s por distancia w_i entre exámenes en un mismo día, donde $i \in [1, 4]$.

El autor agrega cuatro variables más, pero se relacionan a las penalidades por duración de exámenes y salas, que como se mencionó anteriormente se eliminaron para simplificar el problema.

4.2. Constantes

4.2.1. Exámenes

- E : conjunto de exámenes.
- s_i : tamaño del examen $i \in E$.

Se asume que todos los exámenes tienen la misma duración.

4.2.2. Estudiantes

- S : conjunto de estudiantes.
- Rendición de exámenes:

$$t_{is} = \begin{cases} 1 & \text{ssi el estudiante } s \text{ debe rendir el examen } i \\ 0 & \text{en otro caso} \end{cases} \tag{3}$$

4.2.3. Salas

- R : conjunto de salas.
- s_r^R : tamaño de la sala $r \in R$
- w_r^R : un peso que especifica la penalidad por usar la sala r.

Se asume que existen salas infinitas.

4.2.4. Bloques Horarios

Nota: la publicación trata los bloques horarios como periodos, pero para mantener la consistencia de este trabajo se tratarán como bloques horarios.

- P : conjunto de bloques horarios.
- w_p^P : un peso que especifica la penalidad por usar el bloque horario p.
- Exámenes en mismo día:

$$y_{pq} = \begin{cases} 1 & \text{ssi los bloques horarios } p \text{ y } q \text{ pertenecen al mismo día} \\ 0 & \text{en otro caso} \end{cases} \tag{4}$$

4.3. Función Objetivo

$$\text{minimizar} \quad f = \sum_{s \in S} \left(w_0^{16} \cdot C_s^{w_0} + w_1^8 \cdot C_s^{w_1} + w_2^4 \cdot C_s^{w_2} + w_3^2 \cdot C_s^{w_3} + w_4^1 \cdot C_s^{w_4} \right) \tag{5}$$

Puede darse el caso de querer definir la función objetivo para el conjunto completo de estudiantes. En dicho caso, se modifican las constantes:

$$C^{w_0} = \sum_{s \in S} C_s^{w_0} \tag{6}$$

$$C^{w_1} = \sum_{s \in S} C_s^{w_1} \tag{7}$$

$$C^{w_2} = \sum_{s \in S} C_s^{w_2} \tag{8}$$

$$C^{w_3} = \sum_{s \in S} C_s^{w_3} \tag{9}$$

$$C^{w_4} = \sum_{s \in S} C_s^{w_4} \tag{10}$$

$$\tag{11}$$

Lo cual termina generando que la función objetivo sea:

$$\text{minimizar} \quad f = w_0^{16} \cdot C^{w_0} + w_1^8 \cdot C^{w_1} + w_2^4 \cdot C^{w_2} + w_3^2 \cdot C^{w_3} + w_4^1 \cdot C^{w_4} \tag{12}$$

4.4. Restricciones

4.4.1. Restricciones duras

- Cada examen se desarrolla en al menos una sala:

$$\sum_{r \in R} X_{ir}^R \quad \geq \quad 1 \qquad \forall i \in E \tag{13}$$

- Cada examen se desarrolla en al menos un bloque horario:

$$\sum_{p \in P} X_{ip}^P \quad \geq \quad 1 \qquad \forall i \in E \tag{14}$$

- En cualquier bloque horario, cualquier estudiante está rindiendo a lo más un examen:

$$\sum_{i \in E} t_{is} \cdot X_{ip}^P \quad \leq \quad 1 \qquad \forall p \in P, \forall s \in S \tag{15}$$

4.4.2. Restricciones blandas

- Si un estudiante s está inscrito en dos exámenes i y j, y j ocurre:
  - En el bloque horario inmediato después del utilizado por i, la penalización $C_s^{w_0}$ recibe un incremento de 16.
  - En el segundo bloque después del utilizado por i, la penalización $C_s^{w_1}$ recibe un incremento de 8.
  - En el tercer bloque después del utilizado por i, la penalización $C_s^{w_2}$ recibe un incremento de 4.
  - En el cuarto bloque después del utilizado por i, la penalización $C_s^{w_3}$ recibe un incremento de 2.
  - En el quinto bloque después del utilizado por i, la penalización $C_s^{w_4}$ recibe un incremento de 1.

Los items anteriores se pueden resumir en la siguiente restricción, considerando que cada vez que un estudiante registre una penalización, se suma 1 a dicha penalización:

$$C_s^{w_p} = \sum_{\substack{i,j \in E \\ j \neq i}} \sum_{\substack{p,q \in P \\ q=p+1 \ \& \ y_{pq}=1}} t_{is} \cdot t_{js} \cdot X_{ip}^P \cdot X_{jq}^P \tag{16}$$

5. Representación

Mediante la utilización del lenguaje C++, La representación del problema viene dada por dos `structs`: uno de Exámenes (`Exam`) y otro de Estudiantes (`Student`). Para efectos de la resolución del problema, estos structs son utlizados para crear un `unordered_map<int, Exam>` y un `unordered_map<string, Student>`, que se utilizan como una especie de diccionario sin orden que utiliza las `id` como *key* y los `struct` como *value*. Los `structs` y la forma de representarlos se puede entender mejor con el siguiente diagrama:

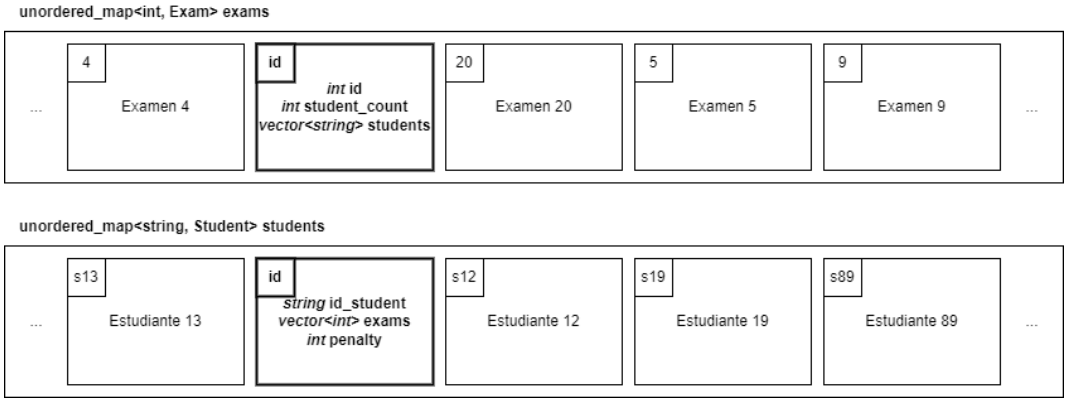

Figura 1: Diagrama que representa (a grandes rasgos) la estructura utlizada.

El uso de `unordered_map` en vez de `map` obedece a poder obtener una mayor capacidad al momento de almacenar los casos de prueba. El `map` convencional permitía un máximo de 10.000 elementos, mientras que el `unordered_map` permite una cantidad limitada solo por la memoria total disponible. Por otro lado, el uso de mapas (independientemente del tipo) permite un mejor tratamiento de los datos al utilizar el mismo `id` como *key* del elemento que se requiere representar.

Para poder representar la planificación horaria, también se recurrió al uso de `unordered_map`, en este caso de tipo `<int, int>`, donde la `key` corresponde al *id* del examen y el *value* al número de bloque horario al que fue asignada.

6. Descripción del algoritmo

El algoritmo utilizado para la resolución del problema fue *Hill Climbing First Improvement* (en adelante, HCFI), técnica incompleta de búsqueda que al recibir una solución inicial y un tipo movimiento, permite la obtención de mejores soluciones a las existentes. La estructura del algoritmo originalmente es de la siguiente forma:

Algorithm 1 HCFI

```
local ← FALSE
s_c ← select a point at random
neighbor ← 0
repeat
    s'_n ← generate a neighbor point in 𝒩(s_c)
    neighbor + +
    if f(s'_n) is better than f(s_c) then
        s_c ← s'_n
        neighbor ← 0
    end if
    if neighbor == max_neighbors then
        local ← TRUE
    end if
until local
```

En el caso de la implementación, la solución inicial viene dada por un Algoritmo *Greedy* (que se describe más adelante), mientras que el movimiento es de tipo *swap*, es decir, intercambia elementos (en este caso, exámenes entre bloques horarios continuos) para buscar una mejor solución. En las siguientes subsecciones se explica más a fondo todo lo anterior.

6.1. Solución Inicial: Algoritmo *Greedy*

El Algoritmo *Greedy* utilizado prioriza la asignación de bloques horarios a aquellos exámenes con mayor cantidad de estudiantes inscritos, evitando en todo momento que ningún estudiante deba rendir dos exámenes al mismo tiempo, sin considerar las penalizaciones por distancia entre los mismos para cada estudiante [9]. Los pasos que el Algoritmo sigue son:

1. Ordenar exámenes de mayor a menor según la cantidad de estudiantes que deben rendirlos.
2. Tomar como **punto de partida** el examen con mayor cantidad de estudiantes que deben rendirlo y asignarlo en el primer bloque horario.
3. Continuar según **función miope**: asignar el siguiente examen con la mayor cantidad de estudiantes que deben rendirlo al siguiente bloque horario que no tenga conflicto entre los estudiantes registrados.

Lo anterior se puede apreciar de mejor manera en la siguiente figura, donde luego de realizar el ordenamiento, se asignan los dos primeros exámenes seguidos porque no pueden ser realizados al mismo tiempo. El tercero puede ser al mismo tiempo que el primero, mientras que el cuarto y quinto no pueden en ninguno de los dos primeros bloques horarios (pero sí pueden ser asignados al mismo tiempo):

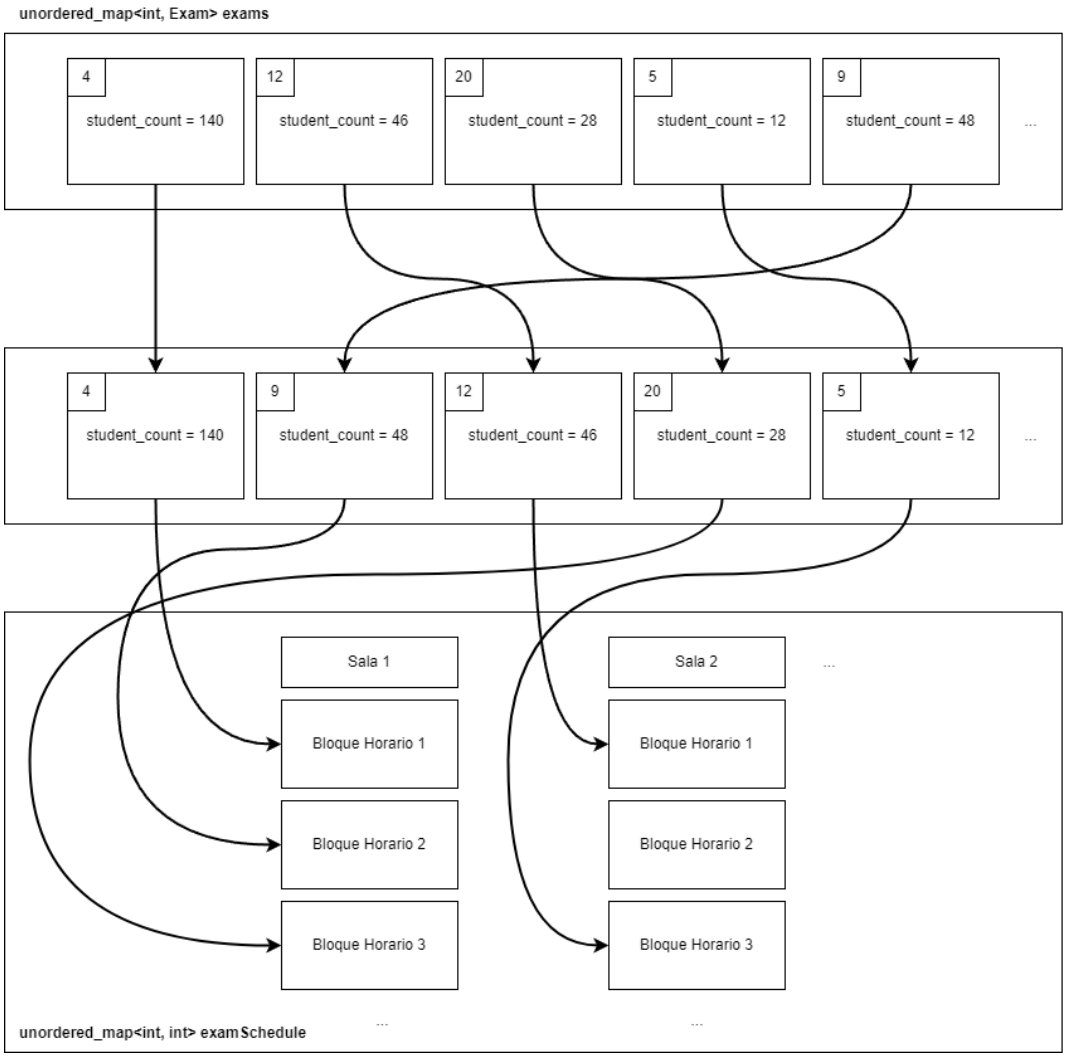

Figura 2: Representación gráfica del Algoritmo *Greedy* implementado.

6.2. Movimiento: Swap

El tipo de movimiento *swap* se encargará de que en cada iteración se pueda proveer de una nueva mejor solución, siempre y cuando esta se encuentre dentro de las restricciones generadas. Cada examen se intercambiará con su examen contiguo para poder buscar una nueva solución [3]. En base al ejemplo de la subsección anterior, a continuación se muestran algunos ejemplos:

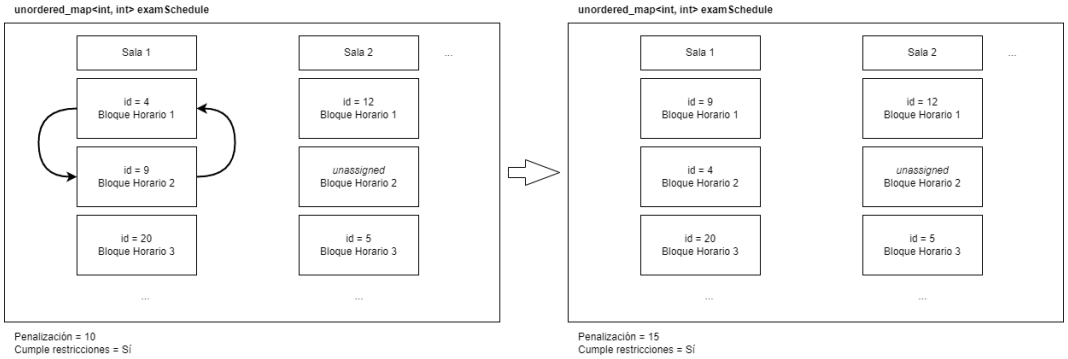

Figura 3: Ejemplo de *swap*.

En el ejemplo anterior, se puede apreciar que si bien el movimiento es válido para la restricciones, la penalización es mayor a la existente, por lo que el movimiento es rechazado. Avanzando en las iteraciones, se tiene el siguiente caso:

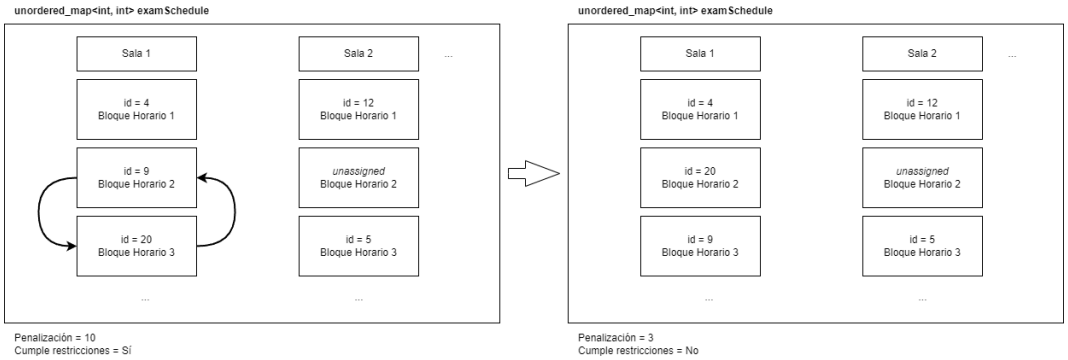

Figura 4: Ejemplo de *swap*.

En este caso, se cumple que la penalización es menor a la existente, pero existen restricciones que no se cumplen, y por ende, se rechaza. Avanzando algunas iteraciones más, se llega a lo siguiente:

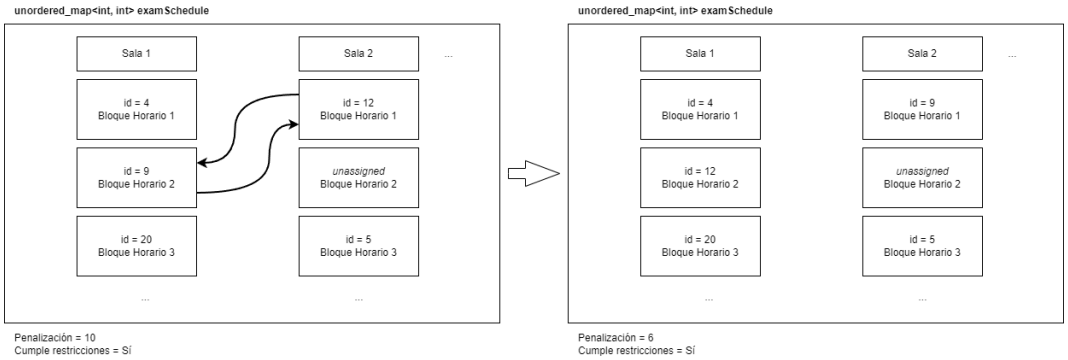

Figura 5: Ejemplo de *swap*.

En este caso, la penalización es menor a la existente y las restricciones se cumplen, por lo que la solución pasa a ser aceptada automáticamente al ser la primera válida.

6.3. HCFI Implementado

Con la solución inicial y el movimiento descritos anteriormente, se puede describir de mejor manera el algoritmo implementado para la solución, que se declara a continuación como pseudocódigo:

Algorithm 2 HCFI Implementado

```
improvement ← FALSE
repeat
    exam1, exam2 ← Exámenes contiguos
    penalizacionActual ← Penalización Greedy
    if hasConflict(exam1, exam2, exams) ≠ FALSE then
        exams ← (exam1 ↔ exam2)
        nuevaPenalizacion ← calculaPenalizacion()
        if nuevaPenalizacion is better than penalizacionActual then
            penalizacionActual ← nuevaPenalizacion
            improvement ← TRUE
        else
            exams ← (exam1 ↔ exam2)
        end if
    end if
until improvement ← TRUE
```

7. Experimentos

Se cuenta con un total de 11 instancias, cada una con una cantidad distinta de exámenes y estudiantes. La finalidad de los experimentos es poder demostrar el funcionamiento de la solución y la mejora de la misma con respecto a lo generado en *Greedy*. Las características de cada instancia se muestran a continuación:

Instancia	Cant. Exámenes	Cant. Estudiantes
Carleton91	682	16 925
TorontoAS92	622	21 266
Carleton92	543	18 419
KingFahd93	461	5 349
LSE91	381	2 726
Trent92	261	4 360
EarlHaig83	190	1 125
TorontoE92	184	2 750
YorkMills83	181	941
St.Andrews83	139	611
EdHEC92	81	2 823

Cuadro 1: Cantidad de exámenes y estudiantes para cada instancia, ordenados según cantidad de exámenes.

La totalidad de las instancias mencionadas anteriormente serán probadas con la finalidad de poder mostrar el dinamismo de tiempos, penalizaciones promedio y cantidad de bloques horarios que se generan según el tamaño de cada una. Todos estos datos, junto con el porcentaje de mejora (*Greedy* vs. HCFI), servirán como indicadores de calidad de la solución.

Para poder estudiar de mejor manera los resultados, primero se realizará una prueba con estas instancias, donde la primera mejora corresponderá a aquella donde la penalización de HCFI sea menor a la de *Greedy*, considerando siempre cumplir con las restricciones. El segundo caso de prueba corresponde a lo mismo, pero la penalización de HCFI debe ser menor a la del 99,5 % de la penalización de *Greedy*. Este caso de prueba es meramente para poder ejecutar una mayor cantidad de iteraciones y hacer notar mejor las diferencias de tiempo entre instancias.

Para realizar los experimentos se utilizó un Asus TUF Gaming F15, con sistema operativo Windows 10 (con WSL de Ubuntu 22.04.4 LTS), procesador Intel Core i5-10300H CPU @ 2.50 GHz y memoria RAM de 16 GB.

8. Resultados

Los resultados se presentan en la siguiente tabla, donde se tiene la cantidad de bloques horarios generados, las penalizaciones promedio de *Greedy* y HCFI, incluyendo el porcentaje de mejora que existe luego de aplicado éste último, la cantidad de iteraciones para lograr la mejora, y el tiempo de ejecución total:

Instancia	Bloques	P.P. *Greedy*	P.P. HCFI	% Mejora	Iteraciones	Tiempo [s]
Carleton91	36	12.036	12.0328	0.027 %	6	4.91789
TorontoAS92	35	8.12663	8.12494	0.021 %	7	5.61402
Carleton92	34	9.6509	9.64917	0.018 %	2	3.88201
KingFahd93	21	40.4221	40.4191	0.007 %	2	2.14748
LSE91	20	26.1654	26.161	0.017 %	2	0.23035
Trent92	22	14.8875	14.8674	0.135 %	3	0.52825
EarlHaig83	25	57.3271	57.2916	0.062 %	2	0.12654
TorontoE92	10	43.4798	43.4674	0.029 %	2	0.41553
YorkMills83	23	48.8555	48.8353	0.041 %	2	0.11675
St.Andrews83	13	122.73	122.704	0.021 %	3	0.12021
EdHEC92	22	25.7407	25.7205	0.078 %	2	0.26308

Cuadro 2: Resultados de experimentos. Notar que P.P. corresponde a Penalización Promedio.

A continuación los resultados aplicando la regla de que la nueva penalización debe ser menor al 99,5 % de la a generada inicialmente por *Greedy*:

Instancia	Bloques	P.P. *Greedy*	P.P. HCFI	% Mejora	Iteraciones	Tiempo [s]
Carleton91	36	12.036	11.9525	0.694 %	184	11.2806
TorontoAS92	35	8.12663	8.07702	0.610 %	145	11.1371
Carleton92	34	9.6509	9.60009	0.526 %	35	5.26365
KingFahd93	21	40.4221	39.8714	1.362 %	270	6.12094
LSE91	20	26.1654	26.0246	0.538 %	154	1.46895
Trent92	22	14.8875	14.7908	0.582 %	26	0.71178
EarlHaig83	25	57.3271	56.9147	0.719 %	66	0.44584
TorontoE92	10	43.4798	42.9385	1.245 %	59	0.85605
YorkMills83	23	48.8555	48.6004	0.522 %	56	0.30909
St.Andrews83	13	122.73	121.525	0.981 %	13	0.14827
EdHEC92	22	25.7407	25.5225	0.848 %	22	0.39182

Cuadro 3: Resultados de experimentos con barrera de Penalización al 99.5 %. Notar que P.P. corresponde a Penalización Promedio.

Lo primero que se puede observar de estos resultados es que el porcentaje de mejora es sumamente bajo, lo cual es consecuente con la regla de HCFI, que busca encontrar la primera solución, sin importar qué tan buena sea: lo importante es que mejore. Por otro lado, se aprecia que los tiempos de ejeución dependen casi exclusivamente de la cantidad de exámenes, lo cual da a entender que la complejidad del problema se expande al haber más rápido al existir una mayor cantidad de exámenes que estudiantes como tal. Las penalizaciones son mayores al existir menos bloques horarios generados, como es el caso de `St.Andrews83`, pero requieren una menor cantidad de iteraciones.

9. Conclusiones

El ETP es un problema de compleja resolución dada su alta cantidad de posibles restricciones dado el contexto de cada institución universitaria. Las técnicas y métodos estudiados permiten resolver este problema con distintas probabilidades de lograr una mejor solución o reducir el costo de las soluciones según sea el caso, pero quedan limitados a la magnitud de las constantes del problema.

La implementación realizada en este trabajo, utilizando *Greedy* y HCFI, permite corroborar la existencia de soluciones eficientes que permiten una mejora, que si bien producto de la naturaleza de HCFI no es la mejor de las soluciones, permite encontrar una leve mejora a lo existente, con rangos de mejora que no pasan del 2 % respecto de lo obtenido en *Greedy*. Por otro lado, se puede destacar que la complejidad de la solución viene dada mayormente por la cantidad de exámenes a registrar y la cantidad de bloques horarios generados, ya que, además de aumentar la cantidad a registrar de estos últimos, también aumenta el tiempo de ejecución para tratar de resolver el problema. Una mayor cantidad de bloques horarios demuestra, en gran parte de los casos, una mayor complejidad del problema.

Para futuras investigaciones, sería interesante seguir explorando nuevas estrategias y enfoques híbridos para la resolución del problema, incluyendo a HCFI como un método absolutamente válido para la resolución del mismo, pero ajustando la barrera de penalización mínima a mejorar.

Referencias

[1] Zahra Naji Azimi. Hybrid heuristics for examination timetabling problem. *Applied Mathematics and Computation*, 163(2):705–733, 2005.

[2] EK Burke, Jeffrey Kingston, and D De Werra. 5.6: Applications to timetabling. *Handbook of graph theory*, 445:4, 2004.

[3] Yuri Bykov and Sanja Petrovic. A step counting hill climbing algorithm applied to university examination timetabling. *Journal of Scheduling*, 19:479–492, 2016.

[4] Michael W Carter. Or practice-a survey of practical applications of examination timetabling algorithms. *Operations research*, 34(2):193–202, 1986.

[5] Michael W Carter, Gilbert Laporte, and Sau Yan Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of the operational research society*, 47:373–383, 1996.

[6] Tim B Cooper and Jeffrey H Kingston. The complexity of timetable construction problems. In *Practice and Theory of Automated Timetabling: First International Conference Edinburgh, UK, August 29–September 1, 1995 Selected Papers 1*, pages 281–295. Springer, 1996.

[7] Luca Di Gaspero, Barry McCollum, and Andrea Schaerf. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). Technical report, Technical Report QUB/IEEE/Tech/ITC2007/CurriculumCTT/v1. 0, Queen's, 2007.

[8] Luca Di Gaspero and Andrea Schaerf. Tabu search techniques for examination timetabling. In *Practice and Theory of Automated Timetabling III: Third International Conference, PATAT 2000 Konstanz, Germany, August 16–18, 2000 Selected Papers 3*, pages 104–117. Springer, 2001.

[9] Dian Kusumawardani, Ahmad Muklason, and Vicha Azthanty Supoyo. Examination timetabling automation and optimization using greedy-simulated annealing hyper-heuristics algorithm. In *2019 12th International Conference on Information & Communication Technology and System (ICTS)*, pages 1–6. IEEE, 2019.

[10] Nuno Leite, Fernando Melício, and Agostinho C Rosa. A fast simulated annealing algorithm for the examination timetabling problem. *Expert Systems with Applications*, 122:137–151, 2019.

[11] Clemens Nothegger, Alfred Mayer, Andreas Chwatal, and Günther R Raidl. Solving the post enrolment course timetabling problem by ant colony optimization. *Annals of Operations Research*, 194:325–339, 2012.

[12] Nelishia Pillay and Wolfgang Banzhaf. An informed genetic algorithm for the examination timetabling problem. *Applied Soft Computing*, 10(2):457–467, 2010.

[13] Rong Qu, Edmund K Burke, Barry McCollum, Liam TG Merlot, and Sau Y Lee. A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling*, 12:55–89, 2009.

[14] Andrea Schaerf. A survey of automated timetabling. *Artificial intelligence review*, 13:87–127, 1999.

[15] Gunther Schmidt and Thomas Ströhlein. Timetable construction–an annotated bibliography. *The Computer Journal*, 23(4):307–316, 1980.

[16] Jonathan M Thompson and Kathryn A Dowsland. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research*, 63:105–128, 1996.

[17] Gert Woumans, Liesje De Boeck, Jeroen Beliën, and Stefan Creemers. A column generation approach for solving the examination-timetabling problem. *European Journal of Operational Research*, 253(1):178–194, 2016.