# Inteligencia Artificial
## Informe Final: Problema
## Examination Timetabling Problem (ETP)

Rodrigo Cayazaya Marín

22 de diciembre de 2021

## Evaluación

| | |
|---|---|
| Mejoras 1ra Entrega (20 %): | _____ |
| Código Fuente (10 %): | _____ |
| Representación (15 %): | _____ |
| Descripción del algoritmo (30 %): | _____ |
| Conclusiones (20 %): | _____ |
| Bibliografía (5 %): | _____ |
| **Nota Final (100)**: | _____ |

**Resumen**

Las instituciones académicas deben organizar el calendario de evaluaciones de modo que no se asignen 2 o más exámenes a un estudiante en el mismo bloque horario, es por esto que surgen los problemas de calendarización de exámenes (ETP). En este escrito se resume el estado del arte de esta problemática, además del modelado e implementación de una solución propuesta que responde al problema planteado inicialmente, además de resolver un inconveniente secundario que corresponde a la maximización de bloques horarios entre exámenes rendidos por un estudiante.

## 1. Introducción

Los problemas de calendarización de exámenes (ETP) buscan calendarizar los exámenes de forma que ningún estudiante deba rendir más de 2 exámenes en un mismo periodo de tiempo, asignando la menor cantidad de bloques de horarios o mejor conocidos como timeslots (para efectos prácticos se utilizará la palabra "timeslots") a los exámenes. Este es un problema recurrente en las instituciones académicas, ya que realizan evaluaciones semestralmente o anualmente.

Este problema es bastante interesante, debido a que solamente cambiando el contexto del problema, se puede convertir en un problema completamente diferente, como es el caso del *Nurse-Scheduling Problem (NCP)* y de otros más que se investigarán en el estado del arte, donde además se compararán resultados de diferentes métodos existentes bajo un mismo dataset de la Universidad de Toronto. También se definirá y modelará una variante del problema al agregar la minimización de una penalización ligada a la holgura de tiempo que existe entre exámenes compartidos por estudiantes.

## 2. Definición del Problema

*Examination Timetabling Problem* es un problema de la categoría *Education Timetabling Problems*, que consiste en organizar las evaluaciones semestrales tipo exámenes a través de timeslots, este tipo de problemas es recurrente en las instituciones académicas, debido a las restricciones que conlleva y a la gran cantidad de estudiantes con las que tratan estos establecimientos.

El objetivo principal es **organizar los exámenes de manera que se minimicen los timeslots utilizados**, esto significa que se distribuirán los exámenes utilizando la menor cantidad de timeslots posibles. En la tabla 1 se puede observar cómo se organizarian 5 exámenes utilizando 5 timeslots.

| | |
|---|---|
| Timeslot 1 | Exámen 1 |
| Timeslot 2 | Exámen 2 |
| Timeslot 3 | Exámen 3 |
| Timeslot 4 | Exámen 4 |
| Timeslot 5 | Exámen 5 |

Tabla 1: Posible solución con 5 exámenes y 5 timeslots.

Al observar la tabla 1 se puede preguntar, ¿por qué no colocar los 5 exámenes en el primer timeslot y así utilizar solamente 1 timeslot? (tal como se muestra en la tabla 2). Esto no es factible, ya que puede causar problemas si al menos un estudiante debe rendir más de un exámen, ya que no podría rendirlos simultáneamente.

| | | | | | |
|---|---|---|---|---|---|
| Timeslot 1 | Exámen 1 | Exámen 2 | Exámen 3 | Exámen 4 | Exámen 5 |

Tabla 2: Posible solución con 5 exámenes y 1 timeslot.

De esta premisa nace la restricción principal del problema: **A ningún estudiante se le asignarán 2 o más exámenes en un mismo timeslot**, esta restricción es conocida coloquialmente como "tope de horario".

Para el siguiente objetivo se debe definir los siguientes conceptos:

**Definición 2.1** (Penalización)**.** La penalización es el valor asociado a la cantidad de timeslots que existe entre un exámen y otro. Este valor asociado se puede observar en la tabla 3. Asimismo, la penalización de un estudiante es la suma de las penalizaciones de todos sus exámenes.

**Definición 2.2** (Penalización promedio por estudiante)**.** La penalización promedio por estudiante es la suma de las penalizaciones de todos estudiantes, dividido en la cantidad estudiantes.

| **Cantidad de timeslots** | **Penalización** |
|---|---|
| 0 | 16 |
| 1 | 8 |
| 2 | 4 |
| 3 | 2 |
| 4 | 1 |
| 5 o más | 0 |

Tabla 3: Penalizaciones dependiendo de la cantidad de timeslots entre exámenes.

Con estas definiciones ahora se puede detallar el objetivo secundario de este problema, el cual consiste en **minimizar la penalización promedio por estudiante**. Este objetivo no es

estrictamente necesario llevarlo a cabo para resolver el problema, pero sí es un objetivo deseable.

El problema que se acaba de definir consiste en uno de la categoría *Education Timetabling Problems*, sin embargo no es el único, ya que existen otros 2 problemas más.

***High Schools Timetabling Problem (HSTP)*** y ***University Course Timetabling Problem (UCTP)*** son problemas similares al problema anteriormente planteado, ya que los 3 problemas deben asignar eventos en timeslots, sin embargo tienen pequeñas variaciones.

El objetivo de HSTP es asignar los horarios de clases anuales a los cursos y a los profesores de manera que no exista un "tope de horario", además existe un objetivo secundario que consta de minimizar los timeslots donde (después de dictar una clase) los profesores part-time se encuentren libres, pero tengan que esperar para dictar una clase (coloquialmente llamado "ventana"). Las restricciones también cambian, ya que se agrega la restricción donde no se debe exceder la capacidad máxima de la sala [15].

El objetivo de UCTP es similar al de HSTP, pero no se asignan horarios de manera anual, sino que de manera semestral, y se aplica individualmente a alumnos y no a un curso de alumnos [15].

Otro problema similar, pero alejado de la categoría *Education Timetabling Problems* es el ***Nurse-Scheduling Problem (NCP)***, el cual trata sobre crear calendarios semanales para los turnos que deben tomar las enfermeras. El objetivo principal es simplemente asignar un horario a las enfermeras sin romper las restricciones y su objetivo secundario es minimizar el coste asignado a las preferencias de todas las enfermeras. Sus restricciones son varias: se debe satisfacer la cantidad de turnos por contrato, tener una cierta cantidad de enfermeras de un grado mayor por cada turno, una enfermera de un grado mayor puede reemplazar a una de un grado menor y una enfermera no puede trabajar de día y de noche la misma semana [2].

Como se puede observar, existe una similitud entre el problema ETP anteriormente definido y el problema NSP, ya que ambos tratan con el problema de asignar eventos en horarios bajo ciertas restricciones.

No solamente existen problemas de calendarización en los trabajos e instituciones, también suceden en las actividades, tal como es el caso del ***Sports Timetabling Problem (STP)***. Su objetivo principal consta de organizar partidos de un torneo en timeslots o "jornadas deportivas"que representan un periodo de tiempo donde todos los equipos juegan (por ejemplo, un fin de semana). Las restricciones vienen dadas en que durante este timeslot todos los equipos deben jugar solamente 1 vez, en cada timeslot los equipos deben jugar con un equipo distinto hasta que todos los equipos hayan jugado entre sí, además los equipos deben jugar una cantidad equitativa de veces en su estadio (conocido coloquialmente como "jugar de local") [24].

Es clara la similitud entre problemas, ya que, a pesar de que no es necesario minimizar una cantidad de un evento, sí es necesario organizar una calendarización de estos, tal como se ha visto en los problemas anteriores.

Por último se explicará el problema ***Train Timetabling Problem (TTP)***, de la categoría *Transportation Timetabling Problem*. El objetivo principal de este problema es planificar el calendario para cada tren, minimizando la distancia entre trenes de un mismo recorrido, cabe destacar que los trenes llevan una prioridad asignada. La restricción viene dada por las reparaciones que puedan existir en las vías, por lo que el tren deberá desviarse (equivalente a una penalización). Así surgiendo un objetivo secundario: maximizar la diferencia entre el valor de la prioridad del tren y la penalización [7].

El objetivo del problema TTP es bastante similar al que se revisó de ETP, ya que ambos comparten la creación de un calendario intentando minimizar un evento.

Tanto ETP, TTP, STP, NSP, UCTP y HSTP son problemas que se encuentran relacionados,

ya que todos intentan crear un calendario en base a restricciones, intentando de alguna u otra manera minimizar o maximizar algún objetivo.

# 3. Estado del Arte

## 3.1. Origen

Los problemas de calendarización (*Timetabling Problems*) no son algo nuevo, el primer registro de este tipo de problemas aplicado a una computadora dicta del **1961** por Appleby sobre la categoría *Education Timetabling Problems*[3]. Tres años más tarde en **1964** Sol Broder implementa un programa para *Examination Timetabling Problem*[4]. Posteriormente, aparecieron las demás categorías como *Transportation Timetabling Problem*, que fue introducido por Pullen en **1967** [19].

## 3.2. Diferentes técnicas

Han existido diferentes técnicas para resolver el problema de calendarización de exámenes, el primero fue utilizado en 1964 llamada la técnica de **"grafo basado en técnicas secuenciales"** [20]. Esta técnica consiste en utilizar el algoritmo de coloreo de grafo: con el objetivo de minimizar la cantidad de colores, cada nodo del grafo lleva un color y no pueden existir nodos conectados que contengan un mismo color. Para resolver un ETP, solamente se deben reemplazar los nodos por exámenes, las aristas (conexiones) por las restricciones duras (aquellas restricciones que se deben cumplir para satisfacer el problema) y los colores asignados a los nodos representan el timeslot asociado al exámen. Cabe destacar que esta técnica no sirve para un problema ETP con restricciones blandas (aquellas restricciones que son deseables, pero no obligatorias) [25].

Otra técnica utilizada es la **técnica basada en restricciones**. Consiste en modelar los exámenes como variables con un dominio finito, donde los valores del dominio representan los timeslots y salas donde se dictarán los exámenes, las variables son asignadas de forma secuencial para construir las soluciones del problema. Si algún valor del dominio no puede ser instanciado, se emplea backtracking para encontrar una solución posible [20].

También existen categorías de técnicas, tal como es el caso de las **técnicas basadas en búsqueda local**. Estas buscan soluciones dentro de un vecindario, la estructura de este vecindario depende de la técnica que se utilice. **Tabu Search** es un de estas técnicas, la cual consiste en explorar el espacio de búsqueda, pero sin re-visitar los recientes movimientos que lleva en un lista (lista tabú). El algoritmo se detiene si encuentra una solución que cumpla con los criterios de detención, si esto no ocurre, continua buscando en otro vecindario, aunque este vecindario sea peor que el anterior [13]. Tabu Search no es la única técnica basada en búsqueda local, también existe la técnica **Simulated Annealing**, que utiliza un área mayor del espacio de búsqueda al principio del proceso, para así aceptar peores movimiento con mayor probabilidad, la cual gradualmente va decreciendo a medida que la búsqueda continua. Hoy en día no se utilizan estas técnicas con un vecindario de forma aleatoria, sino que existen diferentes diseños de vecindarios, ayudando a escapar de que el óptimo sea solamente local, una de estas técnicas es usar una estructura de múltiples vecindarios, lo cual otorga más flexibilidad de navegación del espacio de búsqueda [1].

Las técnicas basadas en algoritmos también son una forma de afrontar este tipo de problemas, uno de estos son los **algoritmos basado en la población**. Uno de estos es llamado **Genetic Algorithms**, los cuales simulan el proceso evolutivo de la naturaleza manipulando y evolucionando poblaciones de soluciones dentro del espacio de búsqueda, apuntando a obtener

mejores resultados al pasar las generaciones. Estos algoritmos trabajan con más de 1 solución a la vez, a diferencia de la categoría anterior que solo encontraba 1 solución y la mejoraba con el tiempo [22]. Una variación de los Genetic Algorithms son los **Memetic Algorithms**, estos se diferencian en que los individuos de una población mejoran durante su vida dentro de una sola generación, sin embargo esto requiere de un gran tiempo de cómputo [18]. El último algoritmo basado en la población es el **Ant Algorithm** que consiste en replicar la forma en que las hormigas encuentran la ruta más corta hacia la comida a través de la feromona que dejan en el camino. En este algoritmo, cada hormiga es usada para construir una solución y la información se mantiene como la feromona, la cual sirve para ayudar a generar soluciones para la siguiente etapa [11].

Una técnica un poco alejada de lo normal, es la **técnica de múltiples criterios**, la cual consiste en utilizar distintas restricciones como criterios, en vez de utilizar la suma de los pesos de una restricción. Esta técnica se aleja de las demás porque apunta en distintas direcciones, es por ello que no se puede comparar con el resto de las técnicas [9].

La última técnica llamada **hyper-heuristics** consiste en desarrollar un enfoque más general y no uno tan dependiente y específico de los parámetros, así se logra desarrollar soluciones que no son para problemas específicos [20].

## 3.3. Comparación entre técnicas

A continuación se realizará una comparación entre las distintas técnicas antes descritas, cabe destacar que la implementación de las técnicas no representan un resultado representativo (pero sí aproximado), ya que una técnica se puede representar por distintos algoritmos, obteniendo resultados distintos. También cabe destacar que la *técnica de múltiples criterios* no será evaluada, ya que su enfoque es distinto al de las demás técnicas.

Los datos utilizados son los famosos datasets de Toronto, estos se pueden visualizar con más detalles en la página: $http://www.cs.nott.ac.uk/pszrq/data.htm$.

*Toronto a* tiene como objetivo minimizar la cantidad de timeslots, *Toronto b* tiene como objetivo minimizar el costo promedio por estudiante, *Toronto c* tiene como objetivo minimizar la cantidad de estudiantes que tengan 2 exámenes el mismo día, *Toronto d* tiene como objetivo minimizar la cantidad de estudiantes que tengan 2 exámenes el mismo día y 2 exámenes de noche, por último *Toronto e* tiene como objetivo minimizar la cantidad de estudiantes con 2 exámenes en timeslots consecutivos.

| **Técnica** | **Implementación** | **Dataset** |
|---|---|---|
| Grafo basado en técnicas secuenciales | Carter [8] | *Toronto a y b* |
| Técnica basada en restricciones | Merlot [17] | *Toronto a,b,c y d* |
| Tabu Search | Di Gaspero [10] | *Toronto b* |
| Simulated Annealing | Burke [6] | *Toronto b y d* |
| Genetic Algorithm | Terashima-Marin [23] | *Toronto e* |
| Memetic Algorithm | Burke [5] | *Toronto c* |
| Ant Algorithm | Dowsland [12] | *Toronto a* |
| Hyper-heuristics | Ross [21] | *Toronto e* |

Tabla 4: Diferentes implementaciones.

En la tabla 4 se pueden observar las implementaciones que se usarán para representar a las

diferentes técnicas y los diferentes datasets con que se probaron.

En las tablas 5, 6 y 7 se encuentran los resultados de las implementaciones en los diferentes datasets. Donde se puede concluir que ciertas técnicas obtuvieron mejores resultados dentro de los datasets, como es el caso de la técnica de *Grafo basado en técnicas secuenciales* que obtuvo el mejor rendimiento para minimizar la cantidad de timeslots, pero un rendimiento mediocre al minimizar el costo promedio por estudiante. Esto es debido a que el grafo está diseñado optimizar el coloreo y, por lo tanto, satisfacer las restricciones entre vecinos directamente conectados y no entre todos los vecinos como es el caso de la técnica *Basada en restricciones*, la cual transforma todos los exámenes en variables con dominio finito y se asignan de forma secuencial, logrando así ser más consciente de las restricciones de todas las variables. Es por esto que obtuvo uno de los mejores rendimientos para minimizar el costo promedio por estudiante. Vale la pena destacar también que la técnica de *Algoritmo genético* obtuvo muy buenos resultados al minimizar la cantidad de estudiantes con 2 exámenes en timeslots consecutivos, esto es debido a la complejidad del objetivo y que el algoritmo genético entrega buenos resultados si se le permite diversificar y tiempo para encontrar los mejores descendientes.

Debido a que no existen tantos datos para el resto de los datasets, no es posible realizar una conclusión significativa de estos.

| **Dataset** | **Carter [8]** | **Merlot [17]** | **Dowsland [12]** |
|---|---|---|---|
| car91 | **28** | 30 | 35 |
| car92 | **28** | 31 | 35 |
| ear83 | **22** | 24 | 24 |
| hec92 | **17** | 18 | 18 |
| kfu93 | **19** | 21 | 20 |
| lse91 | **17** | 18 | 18 |
| rye92 | **21** | 22 | 23 |
| pur93 | **35** | - | - |
| sta83 | **13** | **13** | **13** |
| tre92 | **20** | 21 | 23 |
| uta92 | **32** | **32** | 35 |
| ute92 | **10** | 11 | **10** |
| yor83 | **19** | 23 | 21 |

Tabla 5: Resultados con dataset *Toronto a.*

### 3.4. ETP Hoy

Hoy en día existe tecnología que en 1964 no existía, es por ello que tanto los problemas como las soluciones han evolucionado. El usuario ya no necesita escribir a mano los inputs, sino que ahora todo es digital. Es por ello que una tendencia actual es utilizar *Mixed Integer Programming Models* y el framework *Django* para así desarrollar un *Web Based Decision support system.* Así el usuario solo debe ingresar al sitio web, subir los inputs en un archivo, estos se ingresa a una base de datos para correr los modelos a través de servidores y en menos de 2 minutos desplegar el resultado para el usuario [14].

## 4. Modelo Matemático

A continuación se modelará el problema de ETP explicado en la sección *Definición del Problema.*

| Dataset | Carter [8] | Merlot [17] | Di Gaspero [10] | Burke [6] |
|---|---|---|---|---|
| car91 | 7.1 | 5.1 | 5.7 | **4.8** |
| car92 | 6.2 | 4.3 | - | **4.2** |
| ear83 | 36.4 | **35.1** | 39.4 | 35.4 |
| hec92 | 10.8 | **10.6** | 10.9 | 10.8 |
| kfu93 | 14.0 | **13.5** | - | 13.7 |
| lse91 | 10.5 | 10.5 | 12.6 | **10.4** |
| rye92 | **7.3** | - | - | 8.9 |
| pur93 | - | - | - | - |
| sta83 | 161.5 | **157.3** | 157.4 | 159.1 |
| tre92 | 9.6 | 8.4 | - | **8.3** |
| uta92 | 3.5 | 3.5 | 4.1 | **3.4** |
| ute92 | 25.8 | **25.1** | - | 25.7 |
| yor83 | 41.7 | 37.4 | 39.7 | **36.7** |

Tabla 6: Resultados con dataset *Toronto b*.

| Dataset | Merlot [17] | Burke [5] | Merlot [17] | Burke [6] | Terashima-Marin [23] | Ross [21] |
|---|---|---|---|---|---|---|
| car91 | **158** | 331 | - | - | **130** | 283 |
| car92 | **31** | 81 | 1744 | **1506** | **285** | 542 |
| ear83 | - | - | - | - | **723** | 958 |
| hec92 | - | - | - | - | **154** | 224 |
| kfu93 | **247** | 974 | **1082** | 1321 | **223** | 226 |
| lse91 | - | - | - | - | **221** | 263 |
| rye92 | - | - | - | - | **671** | 832 |
| pur93 | - | - | - | - | - | - |
| sta83 | - | - | - | - | **821** | 1058 |
| tre92 | **0** | 3 | - | - | **586** | 604 |
| uta92 | **334** | 772 | - | - | **594** | 855 |
| ute92 | - | - | - | - | **902** | 967 |
| yor83 | - | - | - | - | **708** | 758 |

Tabla 7: Resultados con dataset *Toronto c, d y e*, de izquierda a derecha.

**Variables:**

$$X_{e,t} = \begin{cases} 1, & \text{Si el exámen } e \text{ se dicta en el timeslot } t. \\ 0, & \text{Caso contrario.} \end{cases} \tag{1}$$

$X_{e,p}$ es la variable que se utilizará en la función objetivo principal, el objetivo es asignar la mayor cantidad de exámenes $e$ a un timeslot $t$ [16].

$$penalty(i) = \begin{cases} 16, & \text{Si penalty(i)*=0} \\ 8, & \text{Si penalty(i)*=1} \\ 4, & \text{Si penalty(i)*=2} \\ 2, & \text{Si penalty(i)*=3} \\ 1, & \text{Si penalty(i)*=4} \\ 0, & \text{Caso contrario.} \end{cases} \tag{2}$$

$$penalty(i)* = C_{e,f,i} * (|p_1 * X_{e,p_1} - p_2 * X_{f,p_2}| - 1) \;\; \forall p_1 \neq p_2 \in P \wedge \forall e \neq f \in E \tag{3}$$

penalty(i)* representa la cantidad de timeslots que existen entre los diferentes timeslots asignados a los exámenes que debe rendir el estudiante $i$.

**Constantes:**

- A = Conjunto de alumnos.
- E = Conjunto de exámenes.
- T = Conjunto de timeslots.
- P = Penalización promedio por estudiante.

$$C_{e,f,i} = \begin{cases} 1, & \text{Si el exámen } e \text{ comparte al estudiante } i \text{ con el exámen } f. \\ 0, & \text{Caso contrario.} \end{cases} \quad C_{e,f,i} \in R^{ExE} \tag{4}$$

$$S_{i,e} = \begin{cases} 1, & \text{Si el estudiante } i \text{ debe rendir el exámen } e. \\ 0, & \text{Caso contrario.} \end{cases} \tag{5}$$

**Dominios:**

- $e \in [1, \#E]$
- $i \in [1, \#A]$
- $t \in [1, \#T]$

**Función objetivo:**

- Principal
  Min T: máx $\vec{X}$

  El objetivo principal es minimizar la cantidad de timeslots (T) a la vez que se maximizan la cantidad de exámenes asignados a timeslots.

- Secundaria
  Min P: $\frac{\sum_{i \in A} \text{penalty(i)}}{\#A}$

  El objetivo secundario es minimizar la penalización promedio por estudiante (P).

**Restricciones:**

1. Ningún estudiante rinde 2 o más exámenes en el mismo timeslot [16].
   $\sum_{e \in E} S_{i,e} * X_{e,t} \leq 1 \quad \forall i \in A \wedge \forall t \in T$
2. Cada exámen está asignado a al menos 1 periodo [16].
   $\sum_{t \in T} X_{e,t} \geq 1 \quad \forall e \in E$

# 5. Representación

El problema se representó utilizando las siguientes variables, constantes, dominios y restricciones:

**Variables:**

$$X_{e,t} = \begin{cases} 1, & \text{Si el exámen } e \text{ se dicta en el timeslot } t. \\ 0, & \text{Caso contrario.} \end{cases} \tag{6}$$

$X_{e,p}$ es la variable que se utilizará en la función objetivo principal, el objetivo es asignar la mayor cantidad de exámenes $e$ a un timeslot $t$ [16].

$$penalty(i) = \begin{cases} 16, & \text{Si penalty(i)*=0} \\ 8, & \text{Si penalty(i)*=1} \\ 4, & \text{Si penalty(i)*=2} \\ 2, & \text{Si penalty(i)*=3} \\ 1, & \text{Si penalty(i)*=4} \\ 0, & \text{Caso contrario.} \end{cases} \tag{7}$$

$$penalty(i)* = C_{e,f,i} * (|p_1 * X_{e,p_1} - p_2 * X_{f,p_2}| - 1) \ \ \forall p_1 \neq p_2 \in P \wedge \forall e \neq f \in E \tag{8}$$

penalty(i)* representa la cantidad de timeslots que existen entre los diferentes timeslots asignados a los exámenes que debe rendir el estudiante $i$.

$$Conf_{e,f,i,t} = \begin{cases} 1, & \text{Si } C_{e,f,i} = 1 \wedge \text{el exámen } e \text{ comparte al timeslot } t \text{ con el exámen } f. \\ 0, & \text{Caso contrario.} \end{cases} \quad Conf_{e,f,i,t} \in R^{ExE} \tag{9}$$

$$Res_{T,X,E} = \begin{cases} MinT : máx, & \text{Si existe una solución para el conjunto de exámenes } E. \\ \emptyset, & \text{Caso contrario.} \end{cases} \quad Res_E \in R^{T^E xTE} \tag{10}$$

**Constantes:**

- A = Conjunto de alumnos.
- E = Conjunto de exámenes.
- T = Conjunto de timeslots.
- P = Penalización promedio por estudiante.

$$C_{e,f,i} = \begin{cases} 1, & \text{Si el exámen } e \text{ comparte al estudiante } i \text{ con el exámen } f. \\ 0, & \text{Caso contrario.} \end{cases} \quad C_{e,f,i} \in R^{ExE} \tag{11}$$

$$S_{i,e} = \begin{cases} 1, & \text{Si el estudiante } i \text{ debe rendir el exámen } e. \\ 0, & \text{Caso contrario.} \end{cases} \tag{12}$$

**Dominios:**

- $e \in [1, \#E]$
- $i \in [1, \#A]$
- $t \in [1, \#T]$

**Restricciones:**

1. Ningún estudiante rinde 2 o más exámenes en el mismo timeslot [16].
   $\sum_{e \in E} S_{i,e} * X_{e,t} \leq 1 \quad \forall i \in A \wedge \forall t \in T$
2. Cada exámen está asignado a al menos 1 periodo [16].
   $\sum_{t \in T} X_{e,t} \geq 1 \quad \forall e \in E$

Esta representación, si bien es similar a la presentada en el Modelo Matemático, consta de dos matrices adicionales:

- Matriz *"Res"* de soluciones factibles: De estas soluciones se escogerá la que tenga la menor penalización promedio por estudiante como solución final, su tamaño viene definido como un límite superior que corresponde a la cantidad máxima de soluciones posibles.
- Matriz *"Conj"* de conflictos: Este será utilizado por el algoritmo *Conflict-based backjumping* explicado más adelante, su tamaño representa una relación 1:1 entre los exámenes.

Las soluciones se representaron en forma de árbol, donde cada nodo representa una instanciación de un examen $e$ a un timeslot $t$, la Figura 1.
Sin embargo, esto representa tanto las soluciones factibles como las infactibles, es por esto que se decidió representar las soluciones como un árbol *"unario"*, similar a un grafo, donde se realiza una búsqueda en profundidad dentro del dominio de las variables hasta encontrar una solución factible, la cual es almacenada en el conjunto *Res*. En la Figura 2 se observa como sería una instanciación de 3 exámenes que se encuentran en conflicto.

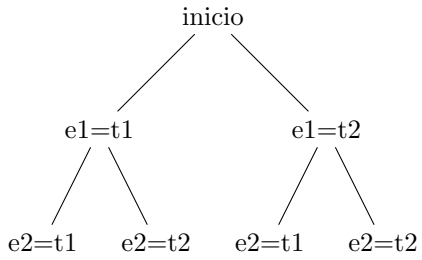


Figura 1: Posible representación en forma de árbol.

Esta representación resulta ser eficiente, ya que reduce el espacio de búsqueda al no buscar entre soluciones infactibles, por lo que ahorra tiempo y almacenamiento. Una vez encontrada una solución, verifica en la siguiente instanciación factible para así no perder soluciones. Esto se realiza para todos los exámenes, así cada solución representa una solución factible del problema original.

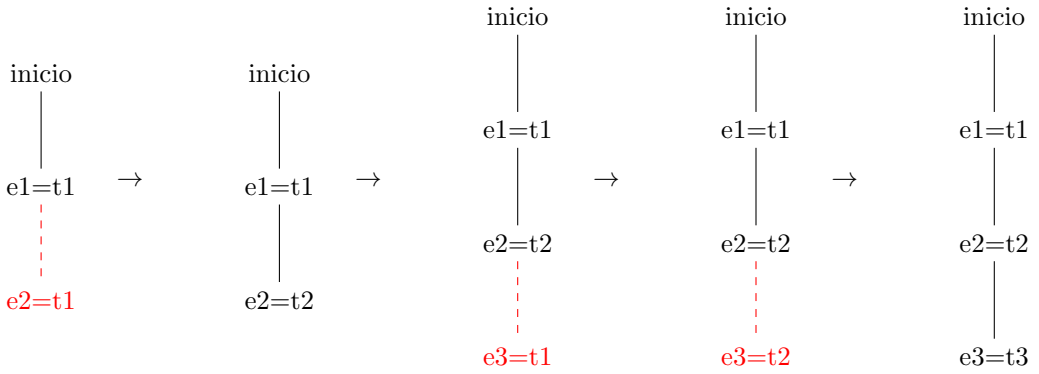


Figura 2: Instanciación de 3 exámenes en conflicto.

Concretamente hablando, las representación del grafo en el lenguaje C se realizaron con una estructura que contiene el id del exámen, el timeslot asociado, un puntero al nodo hijo y uno para el padre. Para el conjunto *Res* se creó una matriz de una estructura que contiene un exámen y su timeslot asociado, así una entrada de la matriz corresponde a un arreglo que contiene una solución, osea, un arreglo de la misma estructura.

# 6. Descripción del algoritmo

Los algoritmos utilizados fueron *Backtracking* y *Conflict-based backjumping (CBJ)*, ambos son algoritmos tipo *Look-Back*. Primero se ingresó una instanciación al árbol, esta instanciación corresponde a un valor del dominio del conjunto de exámenes asignado a un timeslot, así cada exámen se encuentra representado en el árbol, cumpliendo la restricción número 2. Luego se comprueba si es una instanciación factible; si lo es, se mantiene, en caso contrario, se elimina y se realiza el salto atrás. Este salto es llevado a cabo por *Backtracking* o *CBJ* dependiendo del caso. Cabe destacar que, debido a la naturaleza de los grafos en el lenguaje C, tanto las inserciones como las eliminaciones y verificaciones, se realizaron utilizando recursión.

El primer acercamiento que se realizó fue solamente utilizando *Backtracking*, es decir, realizar un salto de tamaño 1 hacia atrás cada vez que no se cumplía la restricción de *tope de horario* (restricción número 1), tal como se puede apreciar en Algorithm 1. Una vez realizado esto, se agregó el salto *CBJ* cuando ningún timeslot podía ser instanciado de forma factible en un exámen (mejor conocido como un *fallo*). Una vez realizado *CBJ*, este se desactiva por toda la rama (hasta la raíz) y se realiza *Backtracking*, esto se realizó para poder encontrar todas las soluciones. Esta implementación se puede apreciar en la sección azul de Algorithm 2.

La matriz *Res* se rellena con soluciones factibles entre el conjunto de exámenes entregado como input y una cantidad fija de timeslots, esta cantidad depende de cuál sea el objetivo del usuario. Si desea buscar la mínima cantidad de timeslots sin un límite inferior, entonces la cantidad fija de timeslots será igual a la cantidad de exámenes y se eliminarán las soluciones con mayor cantidad de timeslots asignados. Por otro lado, si el usuario escoge un límite superior, se agregarán soluciones de una cantidad máxima de timeslots igual a este límite superior. Para ambas implementaciones se calcula y se escoge la solución con menor penalización promedio por estudiante.

**Algorithm 1** Implementación con Backtracking

```
1:  procedure ETP
2:      Res ← Conjunto de soluciones
3:      while existan soluciones factibles do
4:          Arbol ← Árbol vacio
5:          while existan exámenes no instanciados en Arbol do
6:              IdExamen ← Id de un examen no instanciado
7:              while IdExamen no se encuentre en Arbol do
8:                  Timeslot ← Timeslot no instanciado con IdExamen
9:                  Nodo ← instanciacion(IdExamen,Timeslot)
10:                 agregarNodo(Arbol,Nodo)
11:                 if existe tope de horario en Arbol then
12:                     backtracking(Nodo)
13:         agregarSolucion(Res,Arbol)
```

**Algorithm 2** Implementación con Backtracking y CBJ

```
1:  procedure ETP
2:      Res ← Conjunto de soluciones
3:      while existan soluciones factibles do
4:          Arbol ← Árbol vacio
5:          while existan exámenes no instanciados en Arbol do
6:              if es la primera iteración de la rama then
7:                  Conf ← Conjunto de conflicto vacío
8:              IdExamen ← Id de un examen no instanciado
9:              while IdExamen no se encuentre en Arbol do
10:                 Timeslot ← Timeslot no instanciado con IdExamen
11:                 Nodo ← instanciacion(IdExamen,Timeslot)
12:                 agregarNodo(Arbol,Nodo)
13:                 if existe tope de horario en Arbol then
14:                     if no se ha aplicado CBJ en esta rama then
15:                         ExmConf ← Exámen más prematuramente instanciado y en conflicto con Nodo
16:                         agregarConflicto(Conf,ExmConf)
17:                         if hay fallo then
18:                             CBJ(Arbol,Conf)
19:                     if ya se aplicó CBJ en esta rama ó no hay fallo then
20:                         backtracking(Arbol)
21:         agregarSolucion(Res,Arbol)
```

Otro input que se agregó para el usuario es el orden de instanciación de los exámenes que consta de 2 opciones: por orden establecido en el input del archivo *".exm"* (por defecto) u ordenando los exámenes con más conflicto primero.

# 7. Experimentos

Antes de comenzar a experimentar, se deben tener claros los objetivos a alcanzar:

1. [*Principal*] Minimizar la cantidad de timeslots.
2. [*Secundario*] Minimizar la penalización promedio por estudiante.

Para poder cumplir con los objetivos se realizaran 6 instancias por cada dato, en este caso tenemos 3 datos, por lo que obtendremos un total de 18 instancias diferentes. Estas 6 instanciaciones se realizaron cambiando los siguientes parámetros:

1. Ordenado: Orden de instanciación de las variables.
2. Tope: Límite superior de timeslots.

El parámetro de *Ordenado* se instanció para sus 2 opciones: por defecto y por exámenes con más conflicto primero. El parámetro de *Tope* se instanció para aumentar 1 y 2 timeslots de los mínimos encontrados, por ejemplo: si la mínima cantidad de timeslots encontrados fueron 4, entonces el tope se instanciará para 5 y 6. Como se puede observar, para este último parámetro no se respeta la jerarquía de los objetivos, ya que el objetivo *1* pasa a ser secundario y el objetivo *2* se convierte en el objetivo principal.

Los datos utilizados contienen diferentes cantidades de exámenes y estudiantes, así como también diferentes asignaciones entre estudiantes y exámenes. Estas asignaciones se pueden observar en las figuras 3, 4 y 5, siendo los estudiantes graficados como nodos negros con la letra $s$ y los exámenes en nodos azules con la letra $e$.

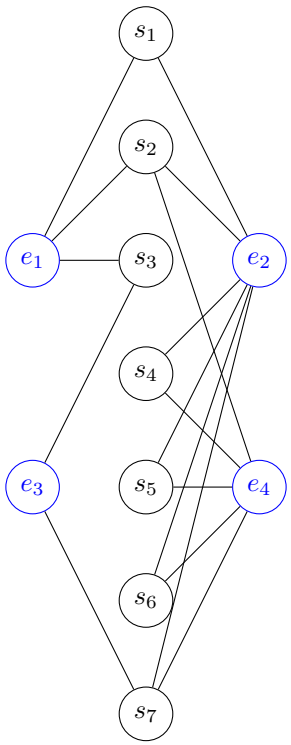


Figura 3: Asignaciones entre exámenes y estudiantes para el conjunto de datos *1*.

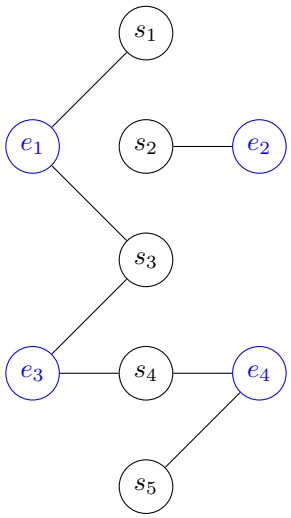


Figura 4: Asignaciones entre exámenes y estudiantes para el conjunto de datos *2*.

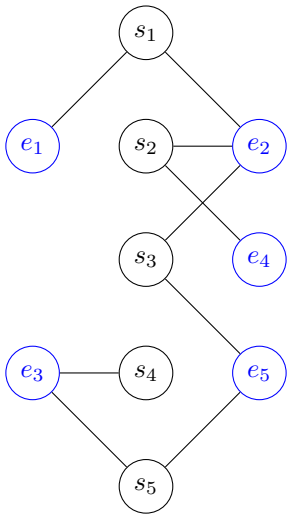


Figura 5: Asignaciones entre exámenes y estudiantes para el conjunto de datos *3*.

El hardware utilizado para los experimentos contiene una CPU *Ryzen 7 5800X* y 16 GB de RAM a una velocidad máxima de 2666MHz. Es importante señalar que, debido a que esta técnica es considerada una *técnica completa*, los resultados fueron deterministas y no se utilizó ningún tipo de aleatoriedad ni semilla.

Cabe destacar que se intentó calcular el tiempo de ejecución de cada instanciación, sin embargo este resultó ser diminuto, por lo cual no lograba almacenarse en una variable de tamaño *double*, esto es debido a los datos limitados de prueba, los cuales son de este tamaño para poder darle un mejor seguimiento y coherencia de los resultados. Es por esto que el tiempo de ejecución no fue considerado como un indicador de calidad, sino que se utilizó la cantidad de *chequeos totales* y la *penalización promedio por estudiante* para ello.

# 8. Resultados

Los resultados de los experimentos se encuentran en las Tablas 8, 9 y 10, los cuales corresponden a los experimentos con el conjunto de datos *1*, *2* y *3*. Cada tabla contiene la cantidad de timeslots que se utlizaron en el resultado final, la penalización promedio por estudiante, la cantidad de chequeos y saltos CBJ que realizó, además de los parámetros modificables como la posibilidad de ordenar las instanciaciones de los exámenes por mayor cantidad de conflictos primero y por un tope como límite superior de timeslots (el valor 0 significa que no existe tope).

| **Timeslots utilizados** | **Penalización** | **Chequeos** | **Saltos CBJ** | **Ordenado** | **Tope** |
|---|---|---|---|---|---|
| 4 | 12.00 | 63 | 0 | No | 0 |
| 4 | 11.14 | 43 | 0 | No | 5 |
| 4 | 8.43 | 51 | 0 | No | 6 |
| 4 | 11.43 | 63 | 0 | Si | 0 |
| 4 | 8.57 | 43 | 0 | Si | 5 |
| 4 | 7.14 | 51 | 0 | Si | 6 |

Tabla 8: Resultados del conjunto de datos *1*.

| **Timeslots utilizados** | **Penalización** | **Chequeos** | **Saltos CBJ** | **Ordenado** | **Tope** |
|---|---|---|---|---|---|
| 2 | 9.60 | 27 | 2 | No | 0 |
| 3 | 8.00 | 23 | 0 | No | 3 |
| 3 | 5.60 | 29 | 0 | No | 4 |
| 2 | 9.60 | 18 | 0 | Si | 0 |
| 3 | 6.40 | 18 | 0 | Si | 3 |
| 3 | 4.80 | 23 | 0 | Si | 4 |

Tabla 9: Resultados del conjunto de datos *2*.

| **Timeslots utilizados** | **Penalización** | **Chequeos** | **Saltos CBJ** | **Ordenado** | **Tope** |
|---|---|---|---|---|---|
| 2 | 12.80 | 45 | 2 | No | 0 |
| 3 | 9.60 | 46 | 0 | No | 3 |
| 3 | 6.40 | 64 | 0 | No | 4 |
| 2 | 12.80 | 23 | 0 | Si | 0 |
| 3 | 8.00 | 41 | 0 | Si | 3 |
| 3 | 5.60 | 55 | 0 | Si | 4 |

Tabla 10: Resultados del conjunto de datos *3*.

De los resultados se puede obtener un análisis bastante interesante, como que no se producen saltos CBJ si la instanciación se encuentra ordenada, esto es debido a que es muy probable que los exámenes que tengan una mayor cantidad de conflictos se encuentren en conflicto entre ellos mismos, entonces si se instancian primero estos exámenes, se evita de manera prematura un posible fallo entre estos. También se puede observar que, como es de esperarse, a mayor cantidad de timeslots utilizados, mayor es la cantidad de chequeos, a menos que se haya realizado un salto CBJ.

Por último, se puede observar que, a pesar de que el tope permita utilizar más timeslots, la cantidad utilizada no varió, esto es debido a que al utilizar la misma cantidad de timeslots, pero separados, genera un mejor desempeño; por ejemplo si el exámen 2 era el exámen con más

conflicto, entonces es muy probable que sea el que se encuentre más alejado de los demás, entonces al ampliar el margen, solamente bastaría con mover este exámen a un timeslot más alejado para poder encontrar el óptimo. De este modo se utilizaría la misma cantidad de timeslots, pero tendría una mejor penalización.

# 9. Conclusiones

De la vasta mayoría de métodos y algoritmos que se presentaron, todos intentan resolver el problema de ETP, pero desde distintas perspectivas. Si bien se puede concluir que algunos métodos son más efectivos que otros, los resultados pueden variar dependiendo del objetivo que se quiere satisfacer o cuántos objetivos se quieren satisfacer (como en el caso de la *técnica de múltiples criterios*). Sin embargo, los resultados obtenidos en las tablas 5, 6 y 7 reflejan las técnicas más prometedoras según sea el objetivo principal:

- Para minimizar la cantidad de timeslots: Grafo basado en técnicas secuenciales.
- Para minimizar el costo promedio por estudiante: Técnica basada en restricciones o Simulated Annealing.
- Para minimizar la cantidad de estudiantes con 2 exámenes el mismo día: Técnica basada en restricciones.
- Para minimizar la cantidad de estudiantes con 2 exámenes el mismo día y 2 exámenes de noche: Técnica basada en restricciones o Simulated Annealing.
- Para minimizar la cantidad de estudiantes con 2 exámenes en timeslots consecutivos: Genetic Algorithm.

Respecto al funcionamiento del algoritmo, las soluciones con menor cantidad de timeslots tienen una penalización promedio por estudiante mayor, es por esto que al poder modificar la cantidad máxima de timeslots se puede analizar de mejor manera el *"trade off"* entre penalización y cantidad de timeslots, generando una ventaja al poder realizar un análisis más profundo como el presentado en la sección anterior.

El hecho de poder ordenar los exámenes por conflicto también produce una ventaja, ya que genera menos cantidades de fallos, logrando así instanciaciones más inteligentes y menos saltos atrás cuando CBJ se encuentra desactivado, tal como se observa en la tabla 9 y 10.

Sin embargo no todo se puede considerar como una ventaja con esta implementación, ya que el algoritmo CBJ genera una matriz de conflicto de tamaño ExE por cada variable, lo cual escala de forma cuadrática (en almacenamiento) correspondiente a la cantidad de exámenes. Además, la falta de un algoritmo Look-Ahead se traduce en instanciaciones poco eficientes, sobre todo cuando se desactiva el algoritmo CBJ.

Este, sin embargo, no es el final para los problemas de ETP. Se siguen desarrollando diferentes implementaciones a las técnicas existentes, tal como es el caso de modificar el vecindario de búsqueda para las *técnicas basadas en búsqueda local*.

Debido a la velocidad en que avanza la tecnología, las soluciones tienen que modernizarse o adaptarse a los nuevos requerimientos. Se propone utilizar la paralelización (como "cloud computing" o "CUDA") para así, gracias a la disminución del trabajo de los servidores y el tiempo de cómputo, poder realizar una *técnica basada en búsqueda local* con distintos vecindarios y quedarse con la que obtenga mejores resultados.

# 10. Bibliografía

# Referencias


[1] S. Abdullah, S. Ahmadi, E K Burke, M. Dror, and B. McCollum. A tabu-based large neighbourhood search methodology for the capacitated examination timetabling problem. *Journal of the Operational Research Society*, 58:1494–1502, 2007.

[2] Uwe Aickelin. An indirect genetic algorithm for a nurse-scheduling problem. *Computers Operations Research*, 31(5):761–778, 2004.

[3] J. S. Appleby, D. V. Blake, and E. A. Newman. Techniques for producing school timetables on a computer and their application to other scheduling problems. *The Computer Journal*, 3(4):237–245, 1961.

[4] Sol Broder. Final examination scheduling. *Commun. ACM*, 7(8):494â–498, 1964.

[5] E. K. Burke, J. P. Newall, and R. F. Weare. A memetic algorithm for university exam timetabling. *Springer Berlin Heidelberg*, pages 241–250, 1996.

[6] EDMUND BURKE, YURI BYKOV, JAMES NEWALL, and SANJA PETROVIC. A time-predefined local search approach to exam timetabling problems. *IIE Transactions*, 36(6):509–528, 2004.

[7] Alberto Caprara, Michele Monaci, Paolo Toth, and Pier Luigi Guida. A lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Applied Mathematics*, 154(5):738–753, 2006.

[8] Gilbert Lee Sau Yan Carter, Michael W. Laporte. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society*, 47(3):373â–383, 1996.

[9] Michael W. Carter and Gilbert Laporte. Recent developments in practical examination timetabling. *Springer Berlin Heidelberg*, pages 1–21, 1996.

[10] Luca Di Gaspero. Recolour, shake and kick: A recipe for the examination timetabling problem. *Proc. of the 4th Int. Conf. on the Practice and Theory of Automated Timetabling*, pages 404–407, 2002.

[11] Marco Dorigo and Christian Blum. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344(2):243–278, 2005.

[12] K A Dowsland and J M Thompson. Ant colony optimization for the examination scheduling problem. *Journal of the Operational Research Society*, 56(4):426–438, 2005.

[13] Michel Gendreau and Jean-Yves Potvin. Tabu search. *Springer US*, pages 165–186, 2005.

[14] Mehmet Güray Güler, Ebru Geçici, Tuğçe Köroğlu, and Emre Becit. A web-based decision support system for examination timetabling. *Expert Systems with Applications*, 183:115363, 2021.

[15] Jeffrey H. Kingston. Educational timetabling. *Automated Scheduling and Planning: From Theory to Practice*, pages 91–108, 2013.

[16] Parkes Andrew J. Burke Edmund K. Qu Rong McCollum Barry, McMullan Paul. A new model for automated examination timetabling. *Annals of Operations Research*, pages 291–315, 2012.

[17] Liam T. G. Merlot, Natashia Boland, Barry D. Hughes, and Peter J. Stuckey. A hybrid algorithm for the examination timetabling problem. *Springer Berlin Heidelberg*, pages 207â–231, 2003.

[18] Pablo Moscato and Michael G Norman. A memetic approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. *Parallel computing and transputer applications*, 1:177–186, 1992.

[19] H. G. M. Pullen and M. H. J. Webb. A computer application to a transport scheduling problem. *The Computer Journal*, 10(1):10–13, 1967.

[20] Burke E. K. McCollum B. Merlot L. T. G. Lee S. Y. Qu, R. A survey of search methodologies and automated system development for examination timetabling. *Journal of Scheduling*, 12(1):1099â–1425, 2009.

[21] P. Ross, J.G. Marin-Blazquez, and E. Hart. Hyper-heuristics applied to class and exam timetabling problems. *Proceedings of the 2004 Congress on Evolutionary Computation*, 2:1691–1698, 2004.

[22] Kumara Sastry, David Goldberg, and Graham Kendall. Genetic algorithms. *Springer US*, pages 97–125, 2005.

[23] Hugo Terashima-Marín, Peter Ross, and Manuel Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, pages 635–642, 1999.

[24] David Van Bulck, Dries Goossens, Jörn Schönberger, and Mario Guajardo. Robinx: A three-field classification and unified data format for round-robin sports timetabling. *European Journal of Operational Research*, 280(2):568–580, 2020.

[25] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85â–86, 1967.