Universidad Politécnica de Victoria.

Ingeniería en Tecnologías de la Información e Innovación Digital.

Materia: Estructura de datos.

Docente: Dr. Said Polanco Martagón.

Unidad I.

Actividad. Avance de proyecto.

Integrantes del equipo.

Jesús Alejandro Aguilar Hernandez. 2430059

Valeria Mayrin Bermudez Raga. 2430215.

Alejandro Sanchez Varela. 2430305

Arturo Rosales Velazquez. 2430152

Ana Zavala Arias. 2430118.

Cd. Victoria, Tamaulipas, a 18 de Octubre de 2025.

Índice.

Introducción.

El Problema de Horarios Universitarios (University Course Timetabling Problem, UCTP) es un desafío logístico central en la gestión académica. Formalmente, se clasifica como un Problema de Optimización Combinatoria (COP), una categoría donde la meta es encontrar la mejor configuración entre un número finito pero vasto de posibilidades.
La esencia del UCTP radica en asignar eventos (cursos, exámenes, laboratorios) a recursos temporales (intervalos de tiempo) y espaciales (aulas, salones) de la manera más eficiente posible, observando un complejo entramado de restricciones y preferencias. La dificultad inherente del UCTP lo posiciona como un problema recurrente y de gran interés dentro de la Investigación Operativa.

Contexto general del University Timetabling Problem.

Concepto e Importancia del Timetabling en la Gestión Universitaria

El timetabling, también conocido como programación de horarios o scheduling, constituye una de las funciones administrativas más críticas en las instituciones de educación superior. Se define como el proceso de asignar recursos educativos (cursos, profesores, estudiantes, aulas) a intervalos de tiempo específicos, satisfaciendo un conjunto de restricciones y optimizando objetivos institucionales.

En el contexto universitario, el timetabling impacta múltiples aspectos operacionales. Una programación adecuada minimiza tiempos de espera, evita conflictos de

recursos y reduce costos operacionales. Permite que los estudiantes accedan a una formación coherente sin conflictos que les impidan cursar materias simultáneamente. Una programación considerada mejora la satisfacción de estudiantes y profesores, reduce fricciones derivadas de cambios de último minuto y optimiza el consumo energético de edificios.

Desde la perspectiva de la investigación operativa, el timetabling se clasifica como un Problema de Optimización Combinatoria (COP) de complejidad NP-completa, lo que significa que no existe un algoritmo conocido capaz de resolverlo en tiempo polinomial para instancias de tamaño realista. Esta característica lo posiciona como un desafío permanente y de alto valor en la academia.

Tipos de Problemas de Horarios

El timetabling presenta variantes especializadas según el contexto educativo. El University Course Timetabling (UCTP) refiere a la asignación de cursos a franjas horarias y aulas durante un período académico regular. Cada evento representa una sesión de clase, con duración típicamente fija, minimizando conflictos entre profesores, estudiantes y aulas.

El Examination Timetabling (ETP) aborda la programación de exámenes, que presenta características distintas al UCTP. Los exámenes son eventos únicos de corta duración, requiriendo coordinación más estricta para evitar que un estudiante presente dos exámenes simultáneamente, enfocándose en garantizar tiempos de preparación entre exámenes consecutivos.

El School Timetabling, aplicable a instituciones preuniversitarias, presenta grupos fijos de estudiantes, horarios más rígidos y consideraciones especiales para materias especializadas. La complejidad es generalmente menor que en universidades. El Staff Scheduling aborda la programación de turnos para personal administrativo, donde la variabilidad de demanda y preferencias personales añaden complejidad.

La variabilidad entre estos tipos implica que las soluciones desarrolladas para uno no son directamente transferibles a otros, aunque comparten principios teóricos fundamentales.

Formulación del problema.

Factores Involucrados: Cursos, Profesores, Aulas, Horarios y Restricciones

El University Timetabling Problem involucra múltiples elementos que interactúan de manera compleja. Una formulación rigurosa requiere definir claramente estos componentes.

Sea $C = \{c_1, c_2, ..., c\square\}$ el conjunto de todos los cursos a programar, donde cada curso posee atributos como duración en horas de clase, número de estudiantes inscritos y requisitos especiales. Sea $P = \{p_1, p_2, ..., p\square\}$ el conjunto de profesores disponibles, donde cada profesor tiene capacidad limitada de horas de docencia, disponibilidad temporal específica y posibles preferencias respecto a horarios. Crucialmente, un profesor no puede impartir más de una clase simultáneamente.

Sea $R = \{r_1, r_2, ..., r\square\}$ el conjunto de recursos espaciales disponibles, donde cada aula posee capacidad máxima de estudiantes, equipamiento específico y disponibilidad. Sea $T = \{t_1, t_2, ..., t\square\}$ el conjunto de intervalos de tiempo disponibles, donde típicamente una semana académica se divide en franjas horarias discretas. Aunque los estudiantes no se programan explícitamente, $S = \{s_1, s_2, ..., s_e\}$ representa a los inscritos, siendo relevantes porque sus horarios generan restricciones de conflicto.

La asignación de horarios se representa mediante la variable de decisión $x_{i\square\square\square} \in \{0, 1\}$, donde i denota el curso, j el profesor, k el aula y l la franja horaria. Si $x_{i\square\square\square} = 1$, el curso i es impartido por el profesor j en el aula k durante la franja l.

Las restricciones duras son obligatorias y su violación hace la solución infactible: un profesor no puede impartir dos cursos simultáneamente, un aula no puede albergar

dos cursos al mismo tiempo, un estudiante no puede asistir a dos clases simultáneamente, el número de estudiantes no puede exceder la capacidad del aula, los cursos se asignan solo a franjas y aulas disponibles, y cada curso debe ser asignado a exactamente un profesor calificado.

Las restricciones blandas son preferencias cuya violación incurre en penalización pero no invalida la solución: algunos profesores prefieren enseñar en horarios específicos, se prefiere evitar concentrar todas las clases de un estudiante en ciertos días, minimizar ventanas libres entre clases, maximizar la ocupación de espacios y considerar preferencias de ubicación respecto a laboratorios o biblioteca.

La solución busca minimizar Z = Σ (violaciones de restricciones duras) × Mpenalidad_dura + Σ (violaciones de restricciones blandas) × w_blanda, donde Mpenalidad_dura es una penalización muy grande que rechaza soluciones infactibles y w_blanda es el peso relativo de cada restricción blanda.

Clasificación del UTP como Problema NP-Difícil

La complejidad computacional del University Timetabling Problem es fundamental para entender por qué requiere técnicas avanzadas de resolución. Un problema pertenece a la clase NP (Nondeterministic Polynomial time) si cualquier solución candidata puede verificarse en tiempo polinomial. El UTP es NP-completo porque una asignación de horarios propuesta puede verificarse en tiempo polinomial mediante evaluación de restricciones, y el problema es tan difícil como otros problemas clásicos NP-completos como Graph Coloring o Satisfiability.

Si P ≠ NP, conjetura ampliamente aceptada, entonces no existe un algoritmo determinista que resuelva el UTP en tiempo polinomial para todas las instancias. El espacio de soluciones crece exponencialmente con el tamaño del problema: para una universidad de tamaño medio, el número de posibles asignaciones puede exceder 10^{500}. Los métodos exactos son inviables para instancias reales.

La solución del UTP en contextos reales depende fundamentalmente de heurísticas y metaheurísticas que sacrifican optimalidad garantizada por tiempo de ejecución razonable. Esta clasificación justifica la investigación de algoritmos como backtracking y búsqueda tabú que buscan soluciones "buenas" en tiempo práctico.

Métodos exactos para hacer el UTP.

La resolución de problemas es el proceso de identificar, analizar y resolver situaciones desafiantes y que también nos permite manejar más eficaz e independientemente futuros problemas. En programación, hay infinidad de problemas, pero también existe una gran variedad de soluciones, entre todas esas lo mejor es encontrar la ideal para nuestro problema. (Cristian, 2010)

Los algoritmos o métodos exactos son aquellos que garantizan que la solución encontrada sea la mejor posible, es decir, encuentran la solución óptima a un problema, esto mediante métodos matemáticos rigurosos.

Dentro de este problema, la utilización de los métodos exactos garantizan una solución óptima aunque solo resuelvan problemas pequeños dentro de un tiempo razonable. Los que se abordarán en esta investigación son el de Fuerza Bruta (brute force) y el de Retroceso (backtracking).

Método de Fuerza Bruta (Brute Force)

El método de Fuerza Bruta es una técnica de solución de problemas que consiste en probar todas las combinaciones o soluciones posibles hasta encontrar la correcta.

Algunas de sus ventajas incluyen:

- Se aplica para resolver la mayoría de los problemas.
- Es un algoritmo simple y fácil de entender.
- Genera algoritmos para problemas importantes como búsqueda, clasificación, coincidencia de cadenas o multiplicación de matrices.
- Su garantía de excelencia global.

Esta universalidad permite su aplicación a cualquier problema de optimización, independientemente de su complejidad o tamaño.

Sin embargo, es un método lento y no tan creativo como otras técnicas de solución de problemas. (Baturu, 2020)

En el contexto del University Timetabling Problem, el método de fuerza bruta busca encontrar la solución óptima mediante la exploración exhaustiva de todas las configuraciones y combinaciones de horarios posibles.
Esto bajo el mecanismo de búsqueda exhaustiva que opera con el principio operativo básico de probar todas las configuraciones concebibles hasta encontrar el resultado más satisfactorio.

El uso del método de Fuerza Bruta es relevante para problemas clasificados como NP-completos, como el de los horarios universitarios, donde el espacio de soluciones aumenta significativamente con el aumento de las variables y restricciones involucradas. (Zaulir et al., 2022)

En el contexto del UTP, esto implica considerar cada posible asignación de cada evento a cada secuencia de tiempo y espacio.
El proceso algorítmico se descompone en tres pasos fundamentales:

- **Generación:** Creación de todas las configuraciones posibles del espacio de soluciones de forma exhaustiva y sin omisiones.
- **Evaluación:** Examen de cada solución candidata generada para verificar su factibilidad (cumplimiento de restricciones duras) y calcular su calidad (penalización de restricciones blandas).
- **Selección:** Identificación de la solución factible que posea el mejor valor en la función objetivo (mínima penalización).

A diferencia de los algoritmos heurísticos que emplean "reglas empíricas" para dirigirse a las soluciones más prometedoras , la Fuerza Bruta procede ciegamente. Esta simplicidad conceptual hace que su costo computacional sea inaceptable para problemas combinatorios de gran escala, como el UTP.

La exploración sistemática en la Fuerza Bruta requiere una generación metódica del espacio de estados, usualmente estructurado como un árbol de búsqueda.
Para cualquier búsqueda exhaustiva, es imperativo incorporar estrategias de poda (pruning) para manejar la intratabilidad del espacio. La poda se basa en el principio de que si una asignación parcial lleva a un estado que se sabe que es inviable o

subóptimo, la búsqueda de ese subárbol puede ser terminada prematuramente. (M. C. Chen et al., 2021)

La lógica detrás de esto es que si una violación dura va a ocurrir, es preferible que suceda lo antes posible en el árbol de búsqueda, lo que permite al algoritmo concentrar el esfuerzo de cómputo en las partes más complejas del problema, maximizando la poda del árbol.

Tipo de Restricción	**Definición y Objetivo**	**Ejemplo en UTP**	**Impacto en Fuerza Bruta**
Restricciones Duras (Hard)	Requisitos obligatorios para alcanzar la factibilidad.	Un profesor no puede dar dos clases al mismo tiempo.	Poda Agresiva: Elimina conjuntos completos de asignaciones inviables.
Restricciones Blandas (Soft)	Preferencias o metas de calidad; su violación se penaliza.	Minimizar el número de bloques libres para los estudiantes.	Función Objetivo: Guía la evaluación para encontrar el óptimo.

La consecuencia directa de la complejidad exponencial es la inviabilidad total de la Fuerza Bruta para instancias de tamaño universitario real. Los algoritmos exactos, que incluyen la Fuerza Bruta, solo están garantizados para proporcionar soluciones óptimas, pero su aplicación se restringe únicamente a problemas de tamaño pequeño. Para una instancia real que involucra cientos de eventos, recursos y restricciones, el tiempo de ejecución de la Fuerza Bruta sería gigantesco, haciendo que la garantía de excelencia no se cumpla y por lo tanto sea irrelevante.

Método de Backtracking.

La búsqueda con retroceso (en inglés, backtracking) es una metaheurística algorítmica diseñada para resolver problemas computacionales, notablemente Problemas de Satisfacción de Restricciones (CSPs, por sus siglas en inglés). Su estrategia fundamental consiste en construir soluciones candidatas de manera incremental, pieza por pieza, y evaluar la validez de la solución parcial en cada paso [1].

El algoritmo explora sistemáticamente las posibles soluciones. Cuando determina que una solución parcial actual no puede ser completada para formar una solución válida (es decir, viola alguna de las restricciones del problema), el algoritmo abandona esa línea de exploración. En ese momento, "retrocede" (realiza el *backtrack*) a la decisión anterior y explora la siguiente alternativa disponible. Este proceso se repite recursivamente hasta encontrar una o todas las soluciones posibles, o hasta que se haya explorado todo el espacio de búsqueda sin éxito [2].

A diferencia de un enfoque de fuerza bruta simple (generar y probar), el backtracking integra la verificación de restricciones en el proceso de generación de candidatos, permitiéndole descartar grandes conjuntos de soluciones inviables de forma temprana.

Representación como un árbol de decisiones

El proceso de backtracking se modela conceptualmente como un árbol de espacio de estados (State Space Tree). Este árbol representa el conjunto de todas las posibles secuencias de decisiones (soluciones parciales) que el algoritmo puede tomar [3].

- **Raíz:** Representa el estado inicial del problema, donde no se ha tomado ninguna decisión (la solución parcial está vacía).
- **Nodos Internos:** Cada nodo v en el árbol representa una solución parcial (una secuencia de decisiones).
- **Aristas (Ramas):** Una arista que conecta un nodo v con un nodo hijo w representa la *extensión* de la solución parcial de v mediante una nueva decisión (por ejemplo, asignar un valor a la siguiente variable).
- **Hojas:** Las hojas del árbol representan soluciones candidatas completas (ya sea válidas o inválidas) o callejones sin salida donde una solución parcial no puede extenderse más.

El algoritmo de backtracking realiza una búsqueda en profundidad (Depth-First Search, DFS) sobre este árbol de espacio de estados. La clave del método es que

el árbol no se genera explícitamente en su totalidad; se construye dinámicamente durante la búsqueda [4].

Retroceso ante violación de restricción.

El mecanismo central del backtracking es la poda (pruning) del árbol de espacio de estados. El algoritmo utiliza una función de acotación (bounding function) o un conjunto de restricciones para evaluar cada nodo (solución parcial) que genera [3].

1. **Nodo Prometedor (Promising):** Un nodo se considera "prometedor" si la solución parcial que representa no viola ninguna restricción y, teóricamente, *podría* extenderse para formar una solución completa válida.
2. **Nodo No Prometedor (Non-promising):** Un nodo se considera "no prometedor" o "muerto" si la solución parcial que representa ya viola una restricción.

Proceso de Retroceso: Cuando el algoritmo, en su búsqueda DFS, llega a un nodo v y determina que no es prometedor (viola una restricción), se produce la poda. El algoritmo no explora ninguno de los descendientes de v. En su lugar, *retrocede* inmediatamente a su nodo padre p y prueba la siguiente arista (la siguiente opción) disponible desde p.

Si el algoritmo retrocede a p y ya ha explorado todas sus ramas (todos sus hijos), retrocede un nivel más arriba, al padre de p, y así sucesivamente. Este mecanismo es el que evita explorar subárboles enteros que garantizadamente no contienen soluciones válidas.

Ventajas Frente a Brute Force: Reducción del Espacio de Búsqueda

La ventaja principal del backtracking es la poda (pruning) del espacio de búsqueda, ya que al encontrar una solución inválida el algoritmo no explora los nodos descendientes de esa solución parcial [5]

El backtracking es exponencialmente más eficiente que la fuerza bruta porque el costo de verificar una solución *parcial* es (generalmente) mucho menor que el de generar una solución *completa*. Al descartar ramas enteras del árbol de decisiones, reduce drásticamente el número de candidatos que deben ser explorados [6].

Limitaciones del Método Backtracking

A pesar de su mejora sobre la fuerza bruta, el backtracking estándar (cronológico) posee limitaciones significativas:

1. **Complejidad de Peor Caso:** En el peor de los casos, si ninguna solución parcial puede ser podada tempranamente (o si la solución se encuentra en la última rama explorada), el backtracking debe explorar el árbol de espacio de estados completo. Su complejidad temporal sigue siendo **exponencial**, a menudo del orden de $O(b^m)$, donde b es el factor de ramificación (número de opciones por variable) y m es la profundidad del árbol (número de variables).

2. **Thrashing (Aplastamiento):** Esta es la limitación más citada del backtracking simple [4]. El "thrashing" describe el fenómeno en el que el algoritmo falla repetidamente por la *misma razón* en diferentes partes del árbol. El backtracking cronológico (simple) retrocede solo un nivel, al padre inmediato, incluso si el conflicto fue causado por una decisión tomada mucho más arriba en el árbol. Esto lleva a una exploración redundante de fallos inevitables.

3. **Dependencia del Orden:** La eficiencia del algoritmo es extremadamente sensible al orden en que se seleccionan las variables para ser instanciadas y al orden en que se prueban los valores para esas variables. Un mal ordenamiento puede llevar a un rendimiento cercano al de la fuerza bruta.

Debido a estas limitaciones, se han desarrollado variantes más sofisticadas, como el *backjumping* (retroceso no cronológico) y el *backtracking dirigido por conflictos* (Conflict-Directed Backjumping, CBJ), así como técnicas de *look-ahead* (como *forward checking* o *AC-3*), que intentan identificar conflictos antes de que ocurran, mejorando la eficiencia de la poda [4][6].

Modelado del University Timetabling Problem en términos de grafos

Introducción a la teoría de grafos

La teoría de grafos es una rama de las matemáticas y la informática que representa relaciones entre elementos a través de nodos (o vértices) y aristas (o conexiones). Cada nodo simboliza una entidad (como una clase, persona o tarea) y cada arista.

- **Nodos o vértices:** representan objetos o entidades (como clases, profesores, tareas, o lugares).
- **Aristas o conexiones:** muestran las relaciones o vínculos entre esos elementos.
- La utilidad de los grafos en problemas de planificacion y optimizacion se da en situaciones donde existen múltiples elementos relacionados, ejemplo:
- **Optimización de rutas:** encontrar el camino más corto entre dos puntos (como en Google Maps).
- **Gestión de proyectos:** identificar tareas críticas o dependencias (técnicas PERT y CPM).
- **Redes sociales:** analizar relaciones, influencia o conexiones entre usuarios.

Representación del UTP mediante grafos

En el Timetabling Problem (UTP), los grafos se utilizan para representar los conflictos que existen entre clases, profesores o grupos, de modo que se puedan generar horarios sin traslapes.

Cada nodo (vértice) representa una clase o curso.
Cada arista (conexión) representa un conflicto entre dos clases (por ejemplo, si comparten el mismo profesor, grupo o aula).

El objetivo del timetabling se traduce en colorear el grafo, lo que significa:

Asignar un color a cada nodo, cada color representa una franja horaria.
Regla: ningún par de nodos conectados (conflictivos) puede tener el mismo color.

De esta forma, se logra una asignación de horarios optimizada y sin choques.

Conclusión.

Este informe exploró el desafío de crear horarios universitarios (UTP), un problema complejo debido a la enorme cantidad de combinaciones posibles entre cursos, profesores y aulas.

Primero, analizamos métodos exactos. Vimos que la Fuerza Bruta, que prueba todas las opciones, es simplemente imposible de usar por el tiempo que requeriría. El Backtracking es más eficiente, ya que descarta grupos de horarios malos, pero aun así se vuelve demasiado lento para un problema de la escala de una universidad.

Frente a esto, el modelado con Teoría de Grafos ofrece una solución mucho más práctica. Este enfoque nos permite crear un "mapa" visual del problema, llamado Grafo de Conflictos.

Este método es muy útil porque nos permite ver claramente dónde están los conflictos y usar algoritmos de coloreo para resolverlos. Aunque el coloreo en sí sigue siendo un reto y le cuesta manejar preferencias (como aulas específicas o clases por la mañana), es la base más sólida.

En resumen, mientras que los métodos exactos como Fuerza Bruta o Backtracking no son viables, la Teoría de Grafos nos da la estructura fundamental para entender y desarrollar soluciones eficientes al complejo rompecabezas de los horarios universitarios.

Referencias.

Contexto general del University Timetabling Problem.

1] Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., & Kendall, G. (2021). A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities. IEEE Access, 9, 106515-106529. https://doi.org/10.1109/ACCESS.2021.3100613

[2] Zaulir, Z. M., Aziz, N. L. A., & Aizam, N. A. H. (2022). General university course timetabling mathematical model to a malaysian public university problem. Malaysian Journal of Fundamental and Applied Sciences, 18(1), 82-94. https://doi.org/10.11113/mjfas.v18n1.2408

[3] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., & Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research, 176(1), 177-192. https://doi.org/10.1016/j.ejor.2005.08.012

[4] Biswas, S., Nusrat, S. A., Sharmin, N., & Rahman, M. (2023). Graph coloring in university timetable scheduling. International Journal of Intelligent Systems and Applications, 15(3), 16-32. https://doi.org/10.5815/ijisa.2023.03.02

Formulación del problema.

[1] Rosa-Rivera, F., Flores, J. J., Puente-Montejano, C. A., & Nava-Muñoz, S. E. (2020). Measuring the complexity of university timetabling instances. Journal of Scheduling, 23(2), 183-208. https://doi.org/10.1007/s10951-020-00641-y

[2] Knuth, D. E. (1975). Estimating the efficiency of backtrack programs. Mathematics of Computation, 29(129), 121-136. https://doi.org/10.1090/S0025-5718-1975-0373371-6

[3] Kumar, V. (1992). Algorithms for Constraint-Satisfaction Problems: A Survey. AI Magazine, 13(1), 32-43. https://doi.org/10.1609/aimag.v13i1.976

[4] Prosser, P. (1993). Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence, 9(3), 268-299. https://doi.org/10.1111/j.1467-8640.1993.tb00310.x

Método de Fuerza Bruta.

[1] Cristian, J. C. (2010). *Algoritmos exactos para el problema del Antibandwidth*. http://hdl.handle.net/10115/4228

[2] Baturu, C. ., & Naufal abdi. (2020). Brute Force Algorithm Implementation Of Dictionary Search . Jurnal Info Sains : Informatika Dan Sains, 10(1), 24–30. https://doi.org/10.54209/infosains.v10i1.29

[3] Zaulir, Z. M., Aziz, N. L. A., & Aizam, N. A. H. (2022). General university course timetabling mathematical model to a malaysian public university problem. Malaysian Journal of Fundamental and Applied Sciences, 18(1), 82-94. https://doi.org/10.11113/mjfas.v18n1.2408

[4] M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar and G. Kendall, "A Survey of University Course Timetabling Problem: Perspectives, Trends and Opportunities," in IEEE Access, vol. 9, pp. 106515-106529, 2021, doi: 10.1109/ACCESS.2021.3100613.

Método de Backtracking

[1] D. E. Knuth, "Estimating the efficiency of backtrack programs," *Mathematics of Computation*, vol. 29, no. 129, pp. 121–136, 1975. DOI: 10.1090/S0025-5718-1975-0373371-6

[2] J. Erickson, "Backtracking," en *Algorithms*, Jeffe E., Ed. University of Illinois, Urbana-Champaign, 2019, cap. 2. https://jeffe.cs.illinois.edu/teaching/algorithms/book/02-backtracking.pdf

[3] A. Levitin, "Backtracking," en *Introduction to the Design and Analysis of Algorithms*, 3.ª ed., Pearson, 2012, cap. 12.1.

[4] V. Kumar, "Algorithms for Constraint-Satisfaction Problems: A Survey," *AI Magazine*, vol. 13, no. 1, p. 32, 1992. DOI: 10.1609/aimag.v13i1.976

[5] L. Ntaimo, "Backtracking Algorithms," en *Integer Programming and Combinatorial Optimization*, Texas A&M University, 2012. https://www.worldscientific.com/doi/10.1142/9781786346766_0003

[6] P. Prosser, "Hybrid Algorithms for the Constraint Satisfaction Problem," *Computational Intelligence*, vol. 9, no. 3, pp. 268–299, 1993. DOI: 10.1111/j.1467-8640.1993.tb00310.x

Modelado del University Timetabling Problem en términos de grafos

[1] GraphEverywhere, E. (2020, 10 marzo). *Teoría de grafos*. GraphEverywhere. https://www.grapheverywhere.com/teoria-de-grafos/

[2] Flores, J. A. V. (s. f.). *3.3. Representación de grafos.* | *REA-Grafos*. https://ecosistema.buap.mx/forms/files/dspace-49/33_representacin_de_grafos.html