TECNOLÓGICO
NACIONAL DE MÉXICO

TAREA 1

Unidad 2

Datos Generales:

INSTITUTO TECNOLOGICO DE CULIACAN
TOPICOS DE INTELIGENCIA ARTIFCIAL
PROFESOR:ZURIEL DATHAN MORA FELIX
HORARIO:12-13

ALUMNO: BRAYANT IVAN GONZALEZ OCHOA

Problema de Programación de Trabajos (JSSP)

El **Job Shop Scheduling Problem (JSSP)** es un problema de optimización combinatoria clasificado como **NP-Hard**, lo que implica que su complejidad crece exponencialmente con el tamaño del problema. Se presenta en entornos de producción donde distintos productos deben procesarse en diversas máquinas siguiendo una secuencia específica de operaciones.

Los sistemas **Job Shop** son comunes en producciones de bajo volumen y alta variedad, donde la programación eficiente es crucial para maximizar la productividad. En un problema con **N productos y M máquinas**, el número de posibles programaciones es **(N!)^M**, lo que hace inviable una solución óptima mediante algoritmos convencionales. Por ello, se emplean enfoques basados en **heurísticas y reglas de despacho** para obtener soluciones aproximadas.

Aplicaciones y Contextos

El JSSP no solo se encuentra en la industria, sino también en sectores como servicios y gastronomía. En cadenas de restaurantes o **dark kitchens**, la programación de insumos y la gestión del personal requieren una planificación eficiente para minimizar costos y optimizar tiempos de entrega.

Cuando se combina el JSSP con la asignación de operarios, la complejidad aumenta. Las estrategias de resolución pueden enfocarse en:

1. **Optimizar el JSSP**, priorizando la programación de los trabajos.
2. **Optimizar la asignación de operarios**, minimizando costos laborales.
3. **Balancear ambos factores**, buscando un equilibrio entre eficiencia y costos.

Métodos de Resolución

1. Métodos Exactos

- **Giffler y Thompson (1960)** generan todas las programaciones posibles y seleccionan la mejor.
- **Branch and Bound**, usado en problemas pequeños (hasta 10x10), mejora la eficiencia con grafos disyuntivos.

2. Metaheurísticas

Para problemas mayores, se usan:

- **Algoritmos Genéticos (AG)**: Evolucionan soluciones candidatas mediante selección y mutación.
- **GRASP (Greedy Randomized Adaptive Search Procedure)**: Combina heurísticas codiciosas con búsqueda local.
- **Path-relinking**: Explora combinaciones entre buenas soluciones.

JSSP con Asignación de Operarios

Cuando se considera la asignación de operarios junto con la programación de trabajos, la solución del problema se vuelve más compleja.

Se pueden utilizar:

- **Optimización lexicográfica**, priorizando la minimización del tiempo total (*makespan*).
- **Programación con restricciones (Constraint Programming)**, combinada con relajación lineal para mejorar la asignación de operarios y optimizar recursos.

Estos enfoques modelan la disponibilidad y habilidades del personal, garantizando una asignación eficiente.

Descripción del problema

Foodology es un startup colombiano que busca crear un grupo de marcas virtuales de restaurantes en América Latina mediante la operación de cocinas ocultas. Las cocinas ocultas son una nueva manera en la que los restaurantes operan 100% vía delivery, preparando sus platos en cocinas ubicadas en distintos puntos de la ciudad y aliándose con plataformas de entrega a domicilio para llevar el producto desde las cocinas a sus clientes. En las cocinas se hace la preparación final de los platos, pues varios de los productos necesarios para cada plato vienen con una preparación inicial desde un centro de producción (CP) el cual distribuye dichos productos a todas las cocinas. Como consecuencia, los procesos que se llevan a cabo en las cocinas requieren de menos operaciones y son mucho más rápidos a la hora de entregar los productos.

Al CP llega semanalmente un pedido que contiene la cantidad total requerida de cada producto. Los productos tienen una secuencia de procesos específica y cada proceso tiene una máquina asociada y un tiempo que puede cambiar según la cantidad de producto que se prepare. El pedido se distribuye a lo largo de cada día de la semana en donde se tienen dos turnos de producción de 8 horas cada uno. Para cada uno de estos turnos se define lo que se denomina la hoja de ruta, la cual indica los productos y cantidades que se van a realizar en el turno y los operarios encargados de realizar cada uno de los productos definidos. A partir de esta hoja de ruta cada operario se encarga de seguir la secuencia de procesos específica que requiere el producto asignado durante el 4 turno. Todo el proceso descrito anteriormente para un turno se ejemplifica en el diagrama de la Ilustración 1.

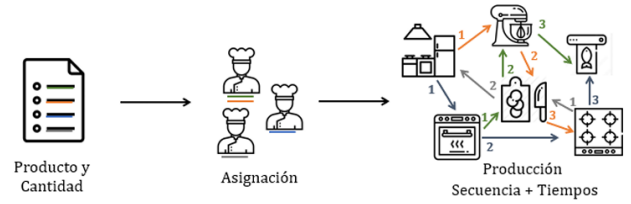


Ilustración 1. Hoja de ruta

metodología

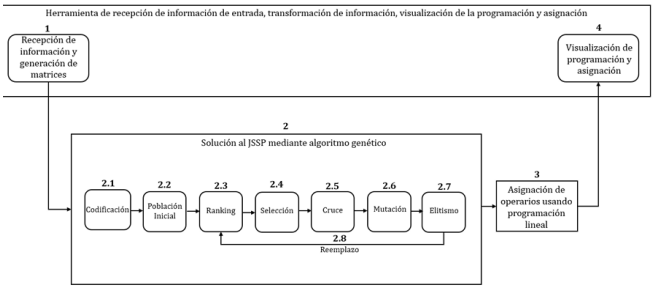


Ilustración 2. Metodología propuesta

Se basa en la estructura propuesta por Artigues et al. (2009) en la cual se prioriza la solución del JSSP frente a la asignación de operarios. Esto se debe a que no se tienen costos de asignación, las habilidades entre operarios son las mismas y la solución del JSSP se asegura que haya una cantidad determinada de operaciones en simultáneo de tal manera que no supere el número de operarios disponibles. Para la solución del JSSP se utiliza una herramienta que recibe como información de entrada los productos a programar y sus respectivas cantidades y la convierte en la secuencia de operaciones que siguen y el tiempo que le toma hacer cada operación según la cantidad del lote.

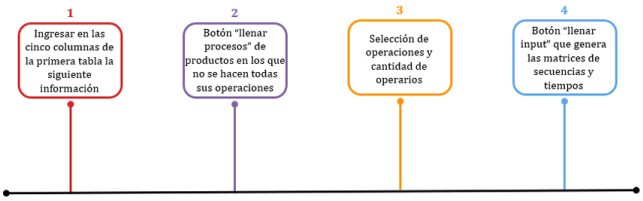


Ilustración 3. Pasos de la herramienta de información de entrada del modelo

Paso 1: Ingresar en las cinco columnas de la primera tabla (Ilustración 4) la siguiente información: Los nombres de los productos. La cantidad para preparar de cada producto. Si para cada producto se quiere realizar el proceso completo o no, es decir todos los pasos de su preparación. Si es un producto que debe ser realizado por un operario de tipo pizzero o no. El operario que tiene una mayor preferencia a realizar cada producto.

Paso 2: Al hacer click sobre el botón "LLENAR PROCESOS" aparecerá en la segunda tabla hacia la derecha de la Ilustración 4 todos los procesos que siguen los productos para los cuales en la columna 3 de la tabla del primer paso se ingresó "NO" como parámetro. Esta segunda tabla consta de tres columnas que indican la selección de procesos que se requieren realizar en el turno, los productos y los procesos.

Paso 3: Se ingresa una "x" en la primera columna de la tabla de procesos (paso 2) para indicar cuáles de estos se quieren programar dentro del turno. Adicionalmente, se ingresa la cantidad de operarios de cada tipo que se tienen disponibles en el turno y el tipo de turno (AM o PM).

Paso 4: Al hacer click sobre el botón "LLENAR INPUT" se construirán dos matrices a partir de la información ingresada en los pasos anteriores: Secuencia de máquinas: El orden de las máquinas que sigue cada producto. Tiempos de procesamiento: El tiempo que se demora cada producto en cada máquina.

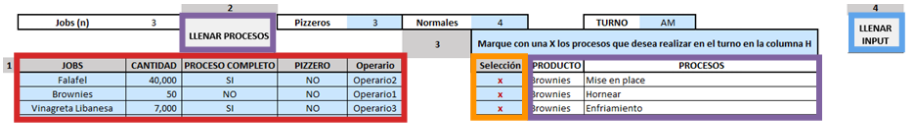


Ilustración 4. Herramienta de información de entrada

Solución al JSSP mediante un algoritmo genético. Un algoritmo genético es una heurística inspirada en la teoría de la evolución natural de Darwin. En este se refleja el proceso de selección natural en el que los mejores individuos son seleccionados para la reproducción y consecuentemente generan una descendencia para las generaciones futuras con mejores características. Para la estructura del algoritmo genético se tienen 8 pasos fundamentales como se muestra en la Ilustración 2, los cuales hacen parte del segundo paso de la metodología propuesta. El primero de ellos establece la población inicial de la cual partirán las generaciones futuras. Los siguientes 5 pasos son los que se encargan de crear nuevas generaciones y por ende son iterativos, es decir que se repiten para cada generación de individuos que

construye el algoritmo. La formulación matemática que representa el modelo del algoritmo genético construido se muestra a continuación.

Conjuntos y Subconjuntos

M: $Máquinas$ (Tiempos de procesamiento Ilustración 7)
N: $Productos$ (Secuencia de máquinas Ilustración 5)
O: $Operaciones$ $\{(n,m,j)\}$. $O = O_{pizzero} \cup O_{regular}$(Secuencia de máquinas Ilustración 5)
E: $Operarios$ (Paso 1 herramienta información de entrada Ilustración 3)
T: $Minutos$ (Duración de la programación)
O_n: $Operaciones\ del\ producto\ n \in N$ (Secuencia de máquinas Ilustración 5)
J_n: $Pasos\ que\ sigue\ el\ producto\ n \in N$ $\{1, \dots, |O_n|\}$ (Secuencia de máquinas Ilustración 5)

Parámetros

p_o: $Duración\ de\ la\ operación\ o \in O$ (Tiempos de procesamiento Ilustración 7)
ρ_t: $Número\ de\ pizzeros\ disponibles\ en\ el\ minuto\ t \in T$ (Paso 1 herramienta información de entrada Ilustración 3)

ϵ_t: $Número\ de\ operarios\ regulares\ disponibles\ en\ el\ minuto\ t \in T$ (Paso 1 herramienta información de entrada Ilustración 3)

Variables de decisión

r_o: $Tiempo\ de\ inicio\ de\ la\ operación\ o \in O$
s_o: $Tiempo\ de\ finalización\ de\ la\ operación\ o \in O$
θ_{ot}: $\begin{cases} 1\ si\ la\ operación\ o \in O\ es\ procesada\ en\ el\ minuto\ t \in T \\ 0\ d.l.c \end{cases}$

Formulación

$$\min TM + \lambda C_{max} \quad (1)$$

s.a,

$$TM = \sum_{n \in N} \sum_{(n,m,j) \in O_n} (s_{(n,m,j)} - s_{(n,m,j-1)}) \quad (2)$$

$$C_{max} \geq c_o, \quad \forall o \in O \quad (3)$$

$$c_o = s_o + p_o, \quad \forall o \in O \quad (4)$$

$$s_{(n,m,j)} \geq c_{(n,m,j-1)}, \quad \forall n \in N, m \in M, j \in J | (n,m,j) \in O, j > 1 \quad (5)$$

$$\left(s_{(n,m,j)} \geq c_{(k,m,l)}\right) \vee \left(s_{(k,m,l)} \geq c_{(n,m,j)}\right), \quad \forall (n,m,j), (k,m,l) \in O \quad (6)$$

$$s_o \geq 0, \quad \forall o \in O \quad (7)$$

$$s_o < (t+1)\pi \wedge s_o > t\pi \implies \theta_{ot} = 1, \quad \forall o \in O, t \in T \quad (8)$$

$$s_o \geq (t+1)\pi \vee c_{nj} \leq t\pi \implies \theta_{ot} = 0, \quad \forall o \in O, t \in T \quad (9)$$

$$\sum_{o \in O_{pizzero}} \theta_{ot} \leq \rho_t, \quad \forall t \in T \quad (10)$$

$$\sum_{o \in O_{regular}} \theta_{ot} \leq \epsilon_t, \quad \forall t \in T \quad (11)$$

$$\sum_{o \in O} \theta_{ot} = 0, \quad \forall t \in T | t > 420, t \leq 480 \quad (12)$$

La función objetivo (1) está compuesta por el total de tiempos muertos de toda la programación más el makespan multiplicado por un λ que pone en la misma escala estas dos medidas. La restricción (2) establece el valor de los tiempos muertos y la restricción (3) el valor del makespan como el máximo tiempo de finalización entre todas las operaciones. Para esto, la restricción (4) establece el tiempo de finalización de cada operación como el tiempo de inicio más el tiempo de procesamiento. Para asegurar la precedencia de las operaciones se tiene la restricción (5) y para evitar superposición de operaciones en una misma máquina la restricción (6). Adicionalmente, se evita tener tiempos de programación negativos mediante la restricción (7) y se establece el valor de (10) (11) (12) (2) la variable binaria que indica si una operación ocurre en un minuto determinado mediante la restricción (8) y (9). Finalmente, se incluyen las restricciones específicas a este problema de Job shop en donde no debe haber más operaciones que operarios disponibles ocurriendo en simultáneo con la (10) y (11) y se asegura que no haya una operación ocurriendo en la hora del almuerzo (12).

Conclusión

El Problema de Programación de Trabajos (JSSP) es uno de los problemas más estudiados en la optimización combinatoria debido a su alta complejidad y su relevancia en la industria manufacturera, la logística y los servicios. Su clasificación como NP-Hard implica que su resolución exacta es inviable para grandes instancias, lo que ha impulsado el desarrollo de métodos heurísticos y metaheurísticos capaces de obtener soluciones eficientes en tiempos razonables.

La importancia del JSSP radica en su aplicabilidad a numerosos sectores productivos donde la asignación eficiente de recursos es clave para la rentabilidad y el rendimiento. En la manufactura, una programación adecuada permite reducir costos operativos y mejorar la eficiencia en el uso de maquinaria y personal. En el sector de servicios, como la organización de eventos o la planificación de turnos de trabajo, una buena programación puede optimizar tiempos y reducir la sobrecarga de empleados. Incluso en la informática, se emplea en la planificación de tareas en sistemas multiprocesador.

A pesar de los avances en la investigación del JSSP, la búsqueda de soluciones óptimas sigue siendo un desafío. Métodos exactos como Branch & Bound o Programación Entera solo son factibles en problemas de pequeña escala, por lo que el uso de metaheurísticas como Algoritmos Genéticos, GRASP y Simulated Annealing ha demostrado ser una alternativa viable. Estas estrategias permiten explorar eficientemente el espacio de soluciones y encontrar configuraciones cercanas al óptimo global en tiempos reducidos.

Un aspecto que complejiza aún más el problema es la inclusión de restricciones adicionales, como la asignación de operarios con distintas habilidades o la variabilidad en los tiempos de procesamiento. En estos casos, se requieren enfoques híbridos que combinen optimización matemática con técnicas basadas en inteligencia artificial para mejorar los resultados.

En conclusión, el Job Shop Scheduling Problem sigue siendo un área de estudio activa, con aplicaciones que abarcan múltiples industrias. A medida que la computación evoluciona, surgen nuevas herramientas y algoritmos que permiten enfrentar estos desafíos con mayor eficacia, impulsando la eficiencia operativa en distintos ámbitos.

Problema de las N Reinas

El **Problema de las N Reinas** es un desafío clásico de optimización y búsqueda en inteligencia artificial y teoría de algoritmos. Consiste en ubicar **N reinas en un tablero de ajedrez de N×N** de manera que ninguna de ellas pueda atacarse entre sí. Esto significa que no pueden compartir la misma fila, columna ni diagonal.

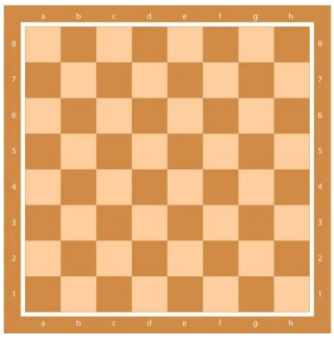
Representación del Problema

Este problema puede formularse como un problema de satisfacción de restricciones (**CSP, Constraint Satisfaction Problem**), donde:

- Cada reina debe ubicarse en una fila y columna únicas.
- No puede haber dos reinas en la misma diagonal.
- El objetivo es encontrar una distribución válida de las reinas en el tablero.

¿De que se trata el problema?

De cuantas maneras se pueden colocar 8 renas en un tablero de 8 x 8 casillas sin que se encuentren entre si.



La reina puede avanzar tantas casillas como quiera de forma lineal ya sea horizontal, vertical o diagonalmente.

Cuentan que el problema se volvió tan popular que incluso el extraordinario matemático Carl Friedrich Gauss trató de resolverlo.

Pero fue Franz Nauck, en 1850, quien enunció la solución: las ocho reinas **se pueden colocar de 92 formas**.

Esa es la primera versión del problema que se generalizó como el problema de las n-reinas y que Simkin le explica a BBC Mundo así:

"Supongamos que n es un número natural, como 1,2,8,100 o un millón. Ahora, imagina un tablero de ajedrez con n filas y n columnas.

¿De cuántas maneras hay de colocar n reinas en el tablero para que **no haya dos que se amenacen** entre sí?

En otras palabras, ¿cuántas formas hay de colocar n reinas en el tablero para que haya una reina en cada fila, una reina en cada columna y no más de una reina en cada diagonal?

El reto lo cautivó.

Le había llegado la hora a las reinas

Para Simkin, el planteamiento tiene "características agradables", se le puede explicar rápidamente a casi cualquier persona, "incluso a los no matemáticos", y eso es algo "inusual" cuando se abordan problemas de ese tipo.

Matemáticamente, una solución puede representarse como un **vector de longitud N**, donde el índice representa la fila y el valor almacenado indica la columna en la que se encuentra la reina.

Complejidad y Naturaleza Computacional

El problema de las N Reinas pertenece a la clase **NP-completo**, lo que significa que su complejidad aumenta exponencialmente con el tamaño del tablero. Para valores pequeños de **N**, se pueden encontrar soluciones de manera eficiente con enfoques de búsqueda sistemática. Sin embargo, para valores grandes de **N**, la cantidad de posibles configuraciones crece de manera factorial, requiriendo métodos más avanzados.

Métodos de Resolución

1. Algoritmos Exactos

- **Backtracking (Vuelta atrás)**: Explora todas las posibles posiciones de las reinas, descartando configuraciones inválidas en cada paso.
- **Branch and Bound (Poda y búsqueda)**: Mejora el backtracking aplicando restricciones que eliminan ramas de la búsqueda antes de explorarlas completamente.

2. Métodos Heurísticos y Metaheurísticas

- **Búsqueda Local con Recocido Simulado**: Modifica soluciones candidatas de forma probabilística para evitar quedarse atrapado en mínimos locales.
- **Algoritmos Genéticos**: Usa una población de soluciones, aplicando operadores evolutivos como mutación y recombinación.
- **Colonia de Hormigas y Enjambre de Partículas**: Modelos bioinspirados que optimizan la colocación de reinas con base en heurísticas inspiradas en la naturaleza.

3. Enfoques Basados en Programación

- **Programación con Restricciones (CSP)**: Define el problema como un conjunto de restricciones a satisfacer, resolviéndolo con técnicas como AC-3 y búsqueda con inferencia.
- **Programación Lineal Entera (ILP)**: Modela el problema con variables binarias y restricciones de no ataque.

Aplicaciones y Extensiones

Más allá del ajedrez, el problema de las N Reinas tiene aplicaciones en:

- **Diseño de circuitos electrónicos** (colocación de componentes sin interferencia).
- **Optimización en redes de comunicación** (asignación de frecuencias sin colisiones).
- **Planeación de recursos y horarios** (distribución eficiente de tareas sin conflictos).

La solución

Simkin calculo que para tableros de ajedrez enormes(n por n casillas) y con muchas reinas, hay alrededor de (0,143n)*n maneras de colocar las reinas sin ninguna amenaza

Supongamos que queremos saber cuántas configuraciones para 1.000.000 de reinas hay", indica el investigador.

Es decir, queremos determinar el número de formas en que se pueden colocar 1.000.000 de reinas en un tablero de 1.000.000 x 1.000.000 (de casillas) sin que se ataquen entre sí.

Para calcular ese número, que es una aproximación, debemos multiplicar 1.000.000 por 0,143 y el resultado, 143.000, lo elevamos a la potencia de 1.000.000.

"En otras palabras, multiplica 143.000 por si mismo un millón de veces. El resultado es un número muy grande, con aproximadamente cinco millones de dígitos".

Ese sería aproximadamente el número de configuraciones.

¿Lo quieres ver con un número más pequeño? Tomemos el 1.000.

En entrevista con BBC Mundo, Jesús Fernando Barbero, matemático e investigador científico del Consejo Superior de Investigaciones Científicas de España, hizo el cálculo usando la ecuación de Simkin.

Si queremos saber aproximadamente cuántas configuraciones hay para 1.000 reinas, donde está la n ponemos 1.000:

0,143 x 1000= 143 y lo elevamos a 1.000.

El docente usó su computadora y este es el resultado que arrojó:

"Me da un número enorme, de más de 2.000 dígitos, pero muy cercano al valor real del número de configuraciones que hay para un tablero de tamaño 1000x1000".

Con la ecuación final de Simkin se llega a un resultado aproximado, no es el número exacto de configuraciones, pero es la cifra más cercana al número real que se puede obtener hasta ahora.

Fuente de la imagen,Getty Images

Él mismo lo indica: "para problemas de este tipo es inusual tener una solución exacta".

Para Barbero, "es un avance enorme" en lo que se refiere al planteamiento de las n-reinas.

"Estos problemas pueden tener enunciados muy simples, pero que luego pueden ser horriblemente complicados".

"El problema estaba atascado, se había conseguido entender cómo resolverlo de manera exacta hasta el número 27".

"Lo que él (Simkin) ha encontrado es un procedimiento para dar una expresión que, aunque no es exacta, comete un error que es pequeño".

"Y eso es satisfactorio: de repente se pasa de no saberse nada sobre el comportamiento del número de soluciones, a ese problema de las reinas para valores grandes, a tener una idea bastante precisa en términos cuantitativos sobre cuántas configuraciones hay para un tablero de tamaño arbitrario".

- La polémica por el problema matemático cuya "demostración impenetrable" casi nadie puede verificar

"En ese sentido, el problema está resuelto, no es una solución final, uno podría aspirar a tener una fórmula exacta que tuviera la propiedad fantástica que al darle el tamaño del tablero me diera el número exacto de configuraciones de reinas que puedo poner".

Pero, aunque es teóricamente posible acercarse a una respuesta mucho más precisa, lo conseguido por Simkin es elogiado por los conocedores.

"Básicamente, lo ha hecho con una precisión que nadie había alcanzado antes", dijo Sean Eberhard, investigador posdoctoral de la Universidad de Cambridge, en un artículo de la revista *Quanta*. Es "de lo más realista que uno puede esperar".

Simkin dice que la razón por la que la solución tomó "tanto tiempo" es porque la misma se basa en los avances recientes que se han conseguido en el campo de las matemáticas conocido como combinatoria probabilística, especialmente en lo que se refiere al análisis de algoritmos informáticos aleatorios.

Conclusión

El problema de las **N Reinas** es un excelente caso de estudio para técnicas de búsqueda y optimización. Su naturaleza combinatoria lo convierte en un referente clave para evaluar algoritmos de inteligencia artificial, heurísticas y metaheurísticas aplicadas a problemas complejos.

El **Problema de las N-Reinas** es un clásico dentro de la teoría de la computación y la optimización combinatoria, ampliamente estudiado por su estructura y sus aplicaciones en diversas áreas de la

informática y la inteligencia artificial. Su enunciado es simple: colocar **N** reinas en un tablero de ajedrez de **N x N** de tal manera que ninguna de ellas pueda atacarse mutuamente. Sin embargo, su resolución no es trivial, ya que el número de configuraciones posibles crece exponencialmente con **N**, lo que lo convierte en un problema de búsqueda combinatoria complejo.

A pesar de su naturaleza aparentemente abstracta, el problema de las N-Reinas tiene aplicaciones prácticas en campos como la planificación de tareas, la asignación de recursos, la generación de patrones sin colisiones en comunicaciones inalámbricas y la inteligencia artificial. En este último ámbito, se emplea para evaluar y comparar el desempeño de algoritmos de búsqueda y optimización.

Los métodos de solución del problema pueden clasificarse en **exactos** y **aproximados**. Los enfoques exactos incluyen **Backtracking**, que explora sistemáticamente todas las configuraciones posibles descartando aquellas que no cumplen con las restricciones; **Programación Entera**, que modela el problema como un sistema de ecuaciones lineales con restricciones; y **Branch & Bound**, que mejora la búsqueda podando ramas del árbol de soluciones cuando se detecta que no pueden conducir a una solución válida. Si bien estos métodos garantizan encontrar todas las soluciones posibles, su costo computacional los hace inviables para valores grandes de **N**.

Por otro lado, las técnicas aproximadas buscan encontrar soluciones de manera más eficiente, sacrificando la garantía de exhaustividad a cambio de tiempos de ejecución reducidos. Entre ellas destacan los **Algoritmos Genéticos**, que evolucionan poblaciones de soluciones mediante mutaciones y selección; **Recocido Simulado**, que explora el espacio de soluciones de manera probabilística evitando quedar atrapado en óptimos locales; y **Colonia de Hormigas**, que simula el comportamiento de estos insectos para construir soluciones válidas de forma iterativa.

A lo largo de los años, el problema de las N-Reinas ha servido como base para la exploración de nuevas técnicas de optimización y búsqueda en espacios combinatorios. Su estudio ha permitido mejorar algoritmos aplicados en problemas reales, como la optimización de redes, la asignación de turnos laborales y la distribución eficiente de tareas en sistemas multiprocesador.

En conclusión, el Problema de las N-Reinas sigue siendo un reto interesante en la informática y la inteligencia artificial, debido a su crecimiento exponencial y a la necesidad de desarrollar algoritmos eficientes para abordarlo. Su simplicidad conceptual lo hace ideal para el estudio de técnicas de optimización, mientras que sus aplicaciones en el mundo real demuestran su relevancia más allá de la teoría matemática. A medida que avanzan los métodos de resolución, este problema continúa siendo una referencia clave en el desarrollo de nuevas estrategias computacionales.

Problema del Árbol de Expansión Mínima (MST)

El **Árbol de Expansión Mínima (MST, Minimum Spanning Tree)** es un problema fundamental en teoría de grafos y optimización combinatoria. Consiste en encontrar un subgrafo de un **grafo no dirigido y conexo**, que:

- **Conecte todos los vértices** sin formar ciclos.
- **Minimice la suma de los pesos** de las aristas incluidas.

El MST tiene aplicaciones en optimización de redes, planificación de infraestructuras y algoritmos de compresión de datos.

Representación del Problema

Dado un grafo **G = (V, E, w)**, donde:

- **V** es el conjunto de **n** vértices.
- **E** es el conjunto de **m** aristas con pesos asociados.
- **w(e)** representa el peso de cada arista **e ∈ E**.

El objetivo es encontrar un subconjunto **T ⊆ E** tal que:

1. **T conecta todos los vértices** sin ciclos (*es un árbol*).
2. **La suma de los pesos de las aristas en T es mínima**.

Complejidad y Naturaleza Computacional

El problema del MST se puede resolver en tiempo **polinómico**, lo que lo hace más eficiente que problemas NP-completos como el **Viajero de Comercio (TSP)** o la **Coloración de Grafos**.

Los algoritmos más eficientes funcionan en **O(m log n)**, aunque con estructuras avanzadas como colas de prioridad con Fibonacci, se puede mejorar a **O(m + n log n)**.

FUNDAMENTOS DE MST
Considere un grafo conectado y no dirigido G = (V, E). Donde V = (v1, v2... vn) es un conjunto finito de vértices (nodos) que pueden representar terminales o estaciones de telecomunicación; y E = { eij e,ij,= (vi,vj), vi,vj, V} en un conjunto finito de enlaces que representan la conexión entre los terminales o estaciones. Cada enlace tiene un número positivo real asociado denotado por W = {wij wij,= w (vi,vj), wij > 0, vi, vj V} representando distancia, costo, etc.

$$T^* = \min \sum_{e_{ij} \in E} W_{ij}$$

Donde T es un conjunto de árboles de expansión del grafo G. El grado de un nodo es el número de enlaces conectados a éste. Un nodo hoja tiene solamente un enlace conectado; de tal manera que el grado de un nodo hoja en un árbol en uno y el de los otros nodos más de uno.

Luego para que un árbol de expansión sea mínimo debe cumplir una de tales condiciones:

a. Ser un subgrafo de G conectado con n-1 enlaces.
b. Ser un subgrafo de G sin ciclos con n-1 enlaces.
c. La sumatoria de los pesos de todos los arcos asociados al subgrafo de G es el menor de todos los subgrafos asociados al grafo G que cumplen con las mismas restricciones.

¿QUÉ ES UN AG?

Son algoritmos de búsqueda basados en la mecánica de la selección natural y de la genética natural. Estos combinan la supervivencia de los individuos más aptos entre las cadenas de estructuras con un

intercambio de información aleatorio para formar un algoritmo de búsqueda. Un árbol de expansión es un mínimo conjunto de enlaces de E que conectan todos los nodos en V y por lo tanto al menos un árbol de expansión puede ser encontrado en un grafo G. El mínimo árbol de expansión denotado por T* es un árbol de expansión cuyo peso total de todos los enlaces es mínimo. Es decir:

Métodos de Resolución

1. Algoritmos Greedy (Voraces)

Aprovechan la propiedad de **subestructura óptima** para construir la solución paso a paso:

- **Algoritmo de Kruskal (1956)**
    - Ordena las aristas por peso y las añade al árbol en orden creciente, evitando ciclos mediante **Conjuntos Disjuntos (Union-Find)**.
    - Complejidad: **O(m log n)** debido a la ordenación.
- **Algoritmo de Prim (1957)**
    - Construye el MST a partir de un vértice inicial, agregando en cada paso la arista más corta que expanda el árbol.
    - Usa colas de prioridad para obtener una complejidad de **O(m log n)** con **montículos binarios** y **O(m + n log n)** con **montículos de Fibonacci**.

2. Algoritmos Basados en Programación

- **Programación Dinámica** *(poco eficiente para grandes instancias)*.
- **Programación Lineal Entera (ILP)** *(formulación con restricciones de conectividad y minimización de costos)*.

PROBLEMA

Elaborar un programa utilizando algoritmos evolutivos para hallar una solución al problema MST que se presenta a continuación (figura 1):

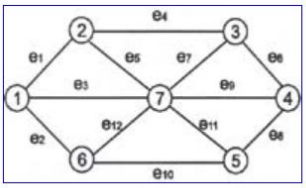


Figura 1. Grafo pesado no dirigido

Las especificaciones y requerimientos de la implementación son los siguientes:

a. Representación genética. Se empleará la denominada Codificación Enlace en donde se asocia un índice k con cada enlace es decir, E = { ek}, k= 1,2, ... k en donde k es el número de enlaces en un nodo. De tal manera que una cadena de bits puede representar una solución candidata indicando cuales enlaces son usados en un árbol de expansión como se ilustra en la figura 2.

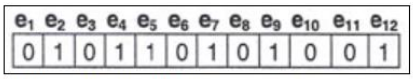


Figura 2. Representación genética de los individuos

b. Cruce. Un punto.
c. Mutación. Realizar mutación como una perturbación aleatoria dentro del rango permisible de enteros entre 1 y n (n = números de nodos en un grafo dado).
d. Selección. Rueda de la ruleta.
e. Parámetros del algoritmo (ver cuadro 1).

Cuadro 1. Parametros de algoritmo

GAP-Generacional	10
Tamaño Poblacional	50
Probabilidad de Cruce	0.5
Probabilidad de mutación	0,01
Generaciones	500
Número de Corridas	30

f. Elaborar un gráfico en donde pueda apreciarse la mejor aptitud en cada corrida del algoritmo.

El algoritmo implementado recibirá un grafo como el mostrado en la figura 1 y determinará el árbol de mínima expansión sabiendo que los costos asociados son como se ilustra en el cuadro 2.

Cuadro 2. Costos asociados al MTS

Nodo	Enlace						
	1	2	3	4	5	6	7
1	0	224	0	0	0	300	539
2	224	0	200	0	0	0	539
3	0	200	0	400	0	0	600
4	0	0	400	0	400	0	200
5	0	0	0	400	0	600	447
6	300	0	0	0	6000	0	283
7	539	539	600	200	447	283	0

En donde cero (0) indica sin conexión directa. Asimismo, en la figura 3 se muestra el resultado de colocar los pesos asociados a cada arco del grafo de la figura 1.

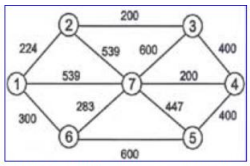


Figura 3. Grafo con pesos asignados a cada arco

Debe ser implementado en C++ Builder, pues presenta una interfaz gráfica donde están predeterminados los parámetros de entrada del algoritmo, estos pueden ser cambiados a conveniencia. Será construido en una forma que permita el proceso de cualquier grafo dado suministrado.

Aplicaciones del MST

El MST tiene aplicaciones en diversas áreas, como:

- **Redes de comunicación** (optimización de cableado y fibra óptica).
- **Diseño de circuitos electrónicos** (mínima cantidad de material conductor).
- **Mapeo y gráficos computacionales** (reducción de bordes en triangulaciones).

- **Compresión de imágenes** (segmentación en árboles de regiones conexas).
- **Algoritmos de clusterización** (como el método de *Single Linkage* en agrupamiento jerárquico).

Conclusión

El Árbol de Expansión Mínima es un problema clásico de optimización combinatoria con soluciones eficientes. Su resolución mediante algoritmos voraces como **Kruskal y Prim** lo hace fundamental en la optimización de redes y estructuras conectadas en múltiples disciplinas.

El problema del Árbol de Expansión Mínima (MST) es una cuestión fundamental en la teoría de grafos y en la optimización de redes. Su importancia radica en su aplicabilidad en una amplia variedad de contextos, desde el diseño de infraestructuras de telecomunicaciones hasta la optimización de rutas en sistemas de transporte y distribución. Dado un grafo conexo y ponderado, el MST busca conectar todos los nodos con un costo mínimo, evitando ciclos y asegurando la conectividad.

Uno de los aspectos más interesantes del MST es que, a diferencia de otros problemas de optimización combinatoria como el Job Shop Scheduling o el TSP, puede resolverse en tiempo polinomial mediante algoritmos eficientes. Entre ellos destacan los algoritmos de **Kruskal** y **Prim**, ambos basados en estrategias de selección de aristas de menor peso para garantizar la solución óptima.

Las aplicaciones del MST son extensas. En el ámbito de las telecomunicaciones, permite diseñar redes de fibra óptica o distribución de electricidad con costos mínimos. En logística, se usa para optimizar rutas de transporte y minimizar el gasto en infraestructura. Incluso en bioinformática, se emplea en la construcción de filogenias para determinar relaciones evolutivas entre especies.

Un aspecto clave en la resolución del MST es su capacidad para adaptarse a distintas variantes del problema. En escenarios donde los costos de conexión varían con el tiempo o donde se imponen restricciones adicionales, pueden aplicarse modificaciones a los algoritmos estándar para obtener soluciones factibles. Por ejemplo, en la planificación de redes inalámbricas, se pueden incluir restricciones de distancia máxima o capacidades de transmisión.

En conclusión, el Árbol de Expansión Mínima es un problema esencial en la optimización de grafos, con aplicaciones en múltiples áreas del conocimiento. Su resolución eficiente mediante algoritmos polinomiales lo convierte en una herramienta poderosa en la toma de decisiones estratégicas. A pesar de su relativa simplicidad en comparación con otros problemas NP-Hard, su importancia práctica y su impacto en la vida cotidiana son innegables.

Problema del Agente Viajero (TSP)

El **Problema del Agente Viajero (TSP, Traveling Salesman Problem)** es un problema de optimización combinatoria ampliamente estudiado en teoría de grafos y logística. Consiste en encontrar el recorrido de costo mínimo que visita un conjunto de ciudades exactamente una vez y regresa a la ciudad de origen.

Es un problema **NP-difícil**, lo que significa que no existe un algoritmo exacto eficiente para grandes instancias. Sin embargo, hay múltiples enfoques heurísticos y metaheurísticos para obtener soluciones aproximadas.

Formulación del Problema

Dado un grafo completo **G = (V, E, w)** donde:

- **V** es el conjunto de **n** ciudades.
- **E** representa las aristas entre todas las ciudades.
- **w(i, j)** indica la distancia o costo de viajar entre las ciudades **i** y **j**.

El objetivo es encontrar un ciclo hamiltoniano de costo mínimo que visite cada ciudad una sola vez y regrese al punto de partida:

min⁡∑i=1nw(xi,xi+1)\min \sum_{i=1}^{n} w(x_i, x_{i+1})mini=1∑nw(xi,xi+1)

sujeto a que cada ciudad sea visitada **una sola vez**.

Complejidad y Dificultad

El **TSP** pertenece a la clase de problemas **NP-hard**, lo que implica que su resolución exacta se vuelve inviable conforme **n** crece. El número de soluciones posibles es **(n - 1)! / 2**, lo que genera una explosión combinatoria.

Para pequeñas instancias **(n ≤ 20)**, los métodos exactos son viables, pero para **n > 1000**, se requieren aproximaciones metaheurísticas.

Métodos de Resolución

1. Métodos Exactos

Son viables para instancias pequeñas:

- **Fuerza bruta**: Evalúa todas las permutaciones posibles **O(n!)**.
- **Programación Dinámica (Held-Karp, 1962)**: Usa el método de subconjuntos con complejidad **$O(n^2 2^n)$**.
- **Branch & Bound**: Reduce el espacio de búsqueda descartando soluciones subóptimas.
- **Programación Lineal Entera (ILP)**: Modelado con desigualdades de subtour.

2. Métodos Aproximados

Para instancias grandes, se emplean heurísticas y metaheurísticas:

- **Algoritmos Greedy** *(voraces)*:
    - *Nearest Neighbor*: Comienza en una ciudad y elige la más cercana.
    - *Christofides (1976)*: Aproximación con factor **1.5** en grafos métricos.
- **Metaheurísticas** *(soluciones de mejor calidad)*:
    - **Recocido Simulado (Simulated Annealing, SA)**: Introduce aleatoriedad para escapar de óptimos locales.

- **Algoritmos Genéticos (GA)**: Evoluciona soluciones mediante mutaciones y recombinaciones.
- **Búsqueda Tabú**: Evita repetir soluciones ya exploradas.
- **Colonia de Hormigas (ACO)**: Modela el problema con feromonas, inspirado en la naturaleza.

3. Aproximaciones Híbridas

Las combinaciones de heurísticas con metaheurísticas logran **mejores tiempos de cómputo y calidad de soluciones**, como:

- **Híbridos ACO + Recocido Simulado** para mejorar convergencia.
- **Enfoques híbridos con aprendizaje profundo**, aplicados en logística y tráfico urbano.

Aplicaciones del TSP

El TSP es un problema con impacto en múltiples áreas:

- **Optimización de rutas** *(logística y distribución de productos)*.
- **Planificación de circuitos impresos** *(minimización de trayectorias de perforación)*.
- **Secuenciación de ADN** *(reconstrucción de fragmentos de genes en biología computacional)*.
- **Diseño de rutas para drones y robots autónomos** *(navegación eficiente)*.
- **Turismo y planificación de viajes** *(optimización de trayectos turísticos)*.

Conclusión

El Problema del Agente Viajero (TSP) es uno de los desafíos más estudiados en la optimización combinatoria debido a su complejidad y a su enorme impacto en sectores clave como la logística, la manufactura y la planificación de rutas. Su clasificación como **NP-Hard** implica que no existe un algoritmo eficiente que garantice la solución óptima en instancias de gran tamaño, lo que ha impulsado la investigación en métodos heurísticos y metaheurísticos para obtener soluciones aproximadas de alta calidad.

El TSP es un problema que modela una situación común en el mundo real: encontrar el recorrido óptimo que minimice costos o tiempos de viaje mientras se visitan una serie de destinos exactamente una vez. Esto lo hace crucial en el diseño de rutas de entrega, la planificación de recorridos turísticos, la gestión de robots autónomos y la optimización de procesos en circuitos electrónicos.

Debido a la explosión combinatoria del número de soluciones posibles, los métodos exactos como Programación Dinámica (Held-Karp) y Branch & Bound solo son viables en instancias pequeñas. Para problemas más grandes, las heurísticas como Nearest Neighbor o Christofides y metaheurísticas como Recocido Simulado, Algoritmos Genéticos y Colonia de Hormigas han demostrado ser herramientas efectivas para encontrar soluciones cercanas al óptimo global.

En los últimos años, la combinación de estos enfoques con técnicas de aprendizaje automático ha permitido mejorar la eficiencia en la resolución del TSP. Modelos de Deep Learning han sido aplicados para generar soluciones aproximadas en tiempo real, lo que abre nuevas posibilidades en la optimización de rutas en escenarios dinámicos.

El TSP no solo es un problema teórico, sino una cuestión de gran impacto en el mundo real. Las empresas de mensajería y transporte lo utilizan para minimizar costos operativos, mientras que en la manufactura se aplica en la optimización de movimientos de maquinaria y herramientas. En la inteligencia artificial y la robótica, el TSP es clave para la navegación eficiente de vehículos autónomos.

En conclusión, el **Problema del Agente Viajero** sigue siendo un área de investigación activa debido a su dificultad y relevancia en la optimización de recursos. Aunque los métodos exactos solo son aplicables en casos pequeños, las heurísticas y metaheurísticas continúan evolucionando para ofrecer soluciones cada vez más eficientes. Su impacto en la vida cotidiana es innegable, haciendo del TSP un problema clave en la ciencia de datos, la logística y la planificación de procesos industriales.

Referencias

Estimating health indicators for thromboembolic, autoimmune and neurological diseases in Colombia between 2009 and 2019

nreinas3.pdf

El problema matemático de las reinas del ajedrez que un científico de Harvard resolvió tras 150 años sin solución - BBC News Mundo

Algoritmo Evolutivo para el Problema de Árbol de Expansión Mínima (MST)

¿Qué es el Travelling Salesman Problem (TSP) y cómo solucionarlo?