# IN7605 Heurísticas para Optimización Entera y Aplicaciones

## Clase 1: Introducción Problemas Discretos y Complejidad

Gonzalo Muñoz, Fernando Ordóñez

18 Agosto, 2025

# Información Administrativa

- **Profesores:** Gonzalo Muñoz (gonzalo.m@uchile.cl), Fernando Ordóñez (fordon@dii.uchile.cl)
- **Auxiliar:** Daniel Hermosilla (daniel.hermosilla.r@ug.uchile.cl)
- **Clases:** Lunes y Miércoles, 16:15 - 17:45 (hora de Chile), online, via link por u-cursos. Argentina: hasta Sept. 3 clase comienza 17:15, desde Sept 8 clase comienza 16:15. Laboratorio en las clases de los miércoles.
- **Evaluaciones:** 2 tareas, 1 proyecto. En grupos entre estudiantes de Argentina y Chile.
- **Notas:** tareas 50%, proyecto 50%.
  **Bibliografía**
  
  - Taillard (2023). *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem*. Springer Cham
  - Berthold, Lodi, Salvagnin (2025). *Primal Heuristics in Integer Programming*. Cambridge University Press
  

# Programa Tentativo

| Lunes | Martes | Miércoles | Notas |
|---|---|---|---|
| 18 Ago:<br>Intro: problemas discretos y complejidad | 19 Ago | 20 Ago:<br>Python, NetworkX<br>TSP y coloreo: heurísticas básicas | |
| 25 Ago:<br>Heurísticas clásicas: greedy, búsqueda local, GRASP | 26 Ago | 27 Ago:<br>GRASP vs greedy en TSP y otros problemas | |
| 1 Sep:<br>Simulated Annealing y Tabu Search | 2 Sep | 3 Sep:<br>SA y TS en TSP, o localización | Tarea 1: Programar una de las heurísticas para un problema aplicado |
| 8 Sep:<br>Heurísticas para ruteo, localización, scheduling | 9 Sep | 10 Sep:<br>Mejoras que exploten estructura de problemas | Cambio de Hora. Clases en Argentina comienzan 16:15 |
| 15 Sep | 16 Sep | 17 Sep | |
| 22 Sep:<br>Problemas clásicos y algoritmos exactos | 23 Sep | 24 Sep:<br>TSP mediante enumeración y en Gurobi | |
| 29 Sep:<br>El paradigma de MILP: relajaciones, redondeo y propagación | 30 Sep | 1 Oct:<br>Callbacks en Gurobi<br>Redondeo y propagación en callbacks | |
| 6 Oct:<br>Local Branching y RINS | 7 Oct | 8 Oct:<br>RINS en callbacks | Distribuir Proyectos<br>Tarea 2: Crear una heurística para un problema aplicado (Santa Claus TSP Challenge?) |

# Programa Tentativo

| Lunes | Martes | Miércoles | Notas |
| --- | --- | --- | --- |
| 13 Oct:<br>Feasibility Pump y Diving | 14 Oct | 15 Oct:<br>Implementación en Gurobi<br>Comparaciones | |
| 20 Oct:<br>Pivoteo y line search | 21 Oct | 22 Oct:<br>Pivoteo desde heurísticas anteriores | Tarea 3: Crear una heurística para MIP genérico (MIP Competition 2023) |
| 27 Oct | 28 Oct | 29 Oct | |
| 3 Nov:<br>MINLP y feasibility pump | 4 Nov | 5 Nov:<br>Relajaciones MINLP y código | |
| 10 Nov:<br>LNS y Undercover | 11 Nov | 12 Nov:<br>Ejemplos en non-convex quadratic (MIP Competition 2025) | |
| 17 Nov:<br>Charla Timo Berthold | 18 Nov | 19 Nov:<br>Revisión de ideas de proyecto | |
| 24 Nov:<br>Presentación de proyectos | 25 Nov | Presentación de proyectos | |

# Definición de Heurística

Wikipedia:

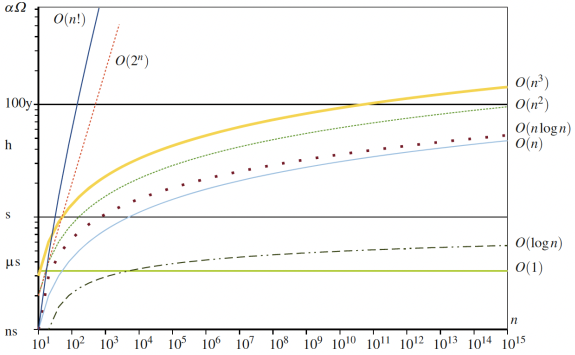
" In mathematical optimization and computer science, heuristic (from Greek eurísko "I find, discover") is a technique designed for problem solving more quickly when classic methods are too slow for finding an exact or approximate solution, or when classic methods fail to find any exact solution in a search space. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.

En optimización: cuando tenemos un problema que es "difícil" de resolver, se pueden utilizar heurísticas que son algoritmos que encuentren "buenas" soluciones ("sin garantías de optimalidad" en general) de forma "rápida".

# Definición de Heurística

- ¿que es un problema "difícil"?
  - en tiempo
  - en complejidad

- ¿que son "buenas" soluciones?
  - benchmark, cotas al problema
  - tiempo de ejecución



# Temario

- Notación de problemas en grafos
- Complejidad y reducción
- Lista de problemas fáciles / difíciles

# Notación de grafos

Por ejemplo, considere el siguiente grafo, o grafo dirigido o digraph, o red:

| Grafos dirigidos | Grafos no dirigidos |
|---|---|
| $G = (N, A)$ | $G = (V, E)$ |
| Nodos $N = \{1, 2, 3, 4\}$ | Vértices $V = \{1, 2, 3, 4\}$ |
| Arcos $A = \{(1,2), (1,3), (3,2), (3,4), (4,1)\}$ | Aristas $E = \{\{1,2\}, \{1,3\}, \{2,3\}, \{3,4\}, \{1,4\}\}$ |

# Notación de grafos

- Una red: es un grafo con valores (enteros) asociados a los vértices y arcos.
  - $c_{ij}$ el costo por unidad de flujo en arco/arista $(i,j)$
  - $u_{ij}$, $l_{ij}$ la capacidad superior e inferior de flujo en arco $(i,j)$
  - $b_i$ valor de oferta/demanda de flujo en nodo $i \in N$.
- Considere la siguiente red para ilustrar las próximas definiciones.

# Definiciones en grafos

- Grafos y redes dirigidos: Un grafo dirigido es un conjunto de nodos $N = \{1, 2, ..., n\}$ y un conjunto de arcos que son pares ordenados de nodos distintos. Por ejemplo para el grafo arriba

  $N =$

  $A =$

  Una red dirigida es un grafo dirigido cuyos nodos y/o arcos tienen valores numéricos asociados (capacidades, costos, y/o ofertas y demandas de flujo).
- Grafos y redes no dirigidos: Un grafo no dirigido es un conjunto de vértices $V = \{1, 2, ..., n\}$ y un conjunto de aristas que son un par de vértices distintos. Las aristas no tienen dirección.

# Definiciones en grafos

- Grado de un nodo/vértice:
  - *indegree* ($\delta_i^+$) es el número de arcos que llegan al nodo
  - *outdegree* ($\delta_i^-$) es el número de arcos que salen del nodo.
  - *grado* es la suma del indegree y outdegree $\delta_i = \delta_i^+ + \delta_i^-$

  Por ejemplo, el nodo 5 del grafo arriba tiene:

  Muestre que $\sum_{i \in N} \delta_i^+ = \sum_{i \in N} \delta_i^- = m$.

- Lista de adyacencia $A(i)$ es la lista de arcos que salen del nodo $i$, es decir $A(i) = \{(i,j) \in A | j \in N\}$. La lista de adyacencia de nodos $A(i) = \{j \in N | (i,j) \in A\}$. Note que $\sum_{i \in N} |A(i)| = m$.

# Definiciones en grafos

- Subgrafo: Un grafo $G_0 = (N_0, A_0)$ donde $N_0 \subseteq N$ y $A_0 \subseteq A$. ¿Cuál es el subgrafo $G_0$ inducido por $N_0 = \{1, 2, 5, 6\}$.

  Dibuje un subgrafo expansor de $G$.

- Paseo, Paseo dirigido (walk): Un paseo es un subgrafo formado por secuencias de nodos y arcos $i_1, a_1, i_2, a_2, ..., a_{r-1}, i_r$ tal que para $1 \leq k \leq r-1$ se tiene que $a_k = (i_k, i_{k+1}) \in A$ o $a_k = (i_{k+1}, i_k) \in A$.
  En un paseo dirigido $a_k = (i_k, i_{k+1}) \in A$ para cada $1 \leq k \leq r-1$. Por ejemplo:

# Definiciones en grafos

- Conexo, Fuertemente conexo: Nodos $i$ y $j$ están conectados si existe un camino de $i$ a $j$. Un grafo es conexo si cualquier par de nodos en el grafo están conectados. Si no es desconectado.
  Un grafo es fuertemente conexo si de cada nodo $i$ hay un camino dirigido a cada nodo $j$ en el grafo.
  Dibuje un grafo conexo que no es fuertemente conexo

  Dibuje un grafo desconectado que tenga un componente fuertemente conexo.

- Corte: Un corte es una partición del conjunto $N$ en dos $S$ y $\overline{S} = N \setminus S$. El conjunto de arcos de $S$ a $\overline{S}$ se denota por $[S, \overline{S}]$.

# Definiciones en grafos

- Árbol, bosque y subarbol: Un árbol es un grafo conexo que no tiene ciclos. Un bosque es un grafo que no contiene ciclos. Puede ser desconectado, pero cada componente es un árbol. Un subarbol es un subgrafo conexo de un árbol.
- **Proposición**
  1. Un árbol con $n$ nodos tiene exactamente $n-1$ arcos
  2. Un árbol tiene al menos dos hojas. (i.e. nodos con grado 1)
  3. Cada par de nodos de un árbol están conectados por un único camino.

- Grafo Bipartito: Un grafo $G=(N,A)$ donde $N$ se particiona en $N_1$ y $N_2$ tal que para todo $(i,j)\in A$ se tiene $i\in N_1$ y $j\in N_2$ o $i\in N_2$ y $j\in N_1$. Dibuje un grafo bipartito

# Árboles expansores de peso mínimo

Dado un grafo $G = (V, E)$ donde $w_e$ es el peso de la arista $e \in E$. Se busca encontrar el subconjunto de arcos $T \subset A$ que sean un árbol, de peso mínimo, que llegue a todos los vértices.

$$\begin{array}{ll} \min & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{s.t.} & x_{ij} \in \{0,1\} \ \ \forall\ (i,j) \in A \\ & x \text{ es un arbol expansor de } G \end{array}$$

Escriba el problema de optimización que corresponde a este problema

# Condiciones de Optimalidad

Dado un árbol expansor $T$ de un grafo $G$. Eliminar un arco $(i, j)$ de $T$ genera un corte, tal que $i \in S$ y $j \in \overline{S}$.

**Theorem**

*(Condiciones de optimalidad de corte) $T^*$ es un MST (minimum cost spanning tree) ssi para todo arco $(i, j)$ en $T^*$, $c_{ij} \leq c_{kl}$ para todo arco $(k, l)$ en corte creado al eliminar $(i, j)$ de $T^*$.*

**dem:**

# Algoritmos para MST

## Algoritmo de Kruskal

Ordene arcos según costos de forma no decreciente

Set $T = \emptyset$

while $|T| < n - 1$ do

  considere el siguiente arco $(i, j)$ en la lista ordenada

  Si $(i, j)$ y $T$ forma un ciclo, se bota $(i, j)$

  else agregar $(i, j)$ a $T$

endwhile

# Algoritmos para MST

## Algoritmo de Prim

Set $S = \{1\}$

while $S \neq N$ do

  encuentre $(i, j) = \mathrm{argmin}\{c_{ij} \mid (i, j) \in [S, \bar{S}]\}$

  agregue $j$ to $S$, set $\mathrm{prior}(j) = i$.

endwhile

# Analisis de complejidad

Vamos a precisar lo de contar el trabajo de un algoritmo.

**Ejemplo** ¿Cuál es el trabajo que realiza el algoritmo de Kruskal?

Assignar valores

Comparaciones

Busqueda de valores

Usaremos algunas simplificaciones para determinar la complejidad de un algoritmo:

- Ignoramos constantes
- Se determinan tiempos de ejecución en terminos de parámetros relevantes del problema
- Se considera las instancias de peor caso
- Todas las operaciones aritmeticas se suponen que toman una operación
- Se supone una *Random Access Machine* (RAM). Se pueden usar arreglos y seleccionar cualquier elemento del arreglo en una operación.
- Todos los números son enteros.

# Analisis de complejidad

Para problemas en redes $G = (N, A)$, los parámetros relevants son:

| | |
|---|---|
| $n := \|N\|$ | número de nodos |
| $m := \|A\|$ | número de arcos |
| $U$ | cota superior en capacidades de flujo en arcos |
| $C$ | cota superior en coeficientes de costo |

Un algoritmo resuelve un problema en red en tiempo

- **pseudo-polinomial** iteraciones son acotadas por un polinomio en $n$, $m$, $U$, $C$.
- **polinomial** iteraciones son acotadas por un polinomio en $n, m, \log U$, and $\log C$.
- **fuertemente polinomial** iteraciones son acotadas por un polinomio en $n$ and $m$.

# Algoritmo de Breadth First Search

Input: Un grafo dirigido $G = (N, A)$, un nodo de partida $s \in N$.
Output: El conjunto de nodos que se pueden visitar de $s$
Definir Arreglos: Reachable[$n$], Prior[$n$].

Set Reachable[$s$] = yes y Reachable[$j$] = no para todo $j \in N - \{s\}$. LIST = {s}
while LIST != empty do
    seleccione $f$ el primer nodo de LIST
    encuentre $j \in A(f)$ tal que Reachable[$j$] == no
    si $j$ existe entonces
        Reachable[$j$] = yes, Prior[$j$] = $f$, agregar $j$ a LIST.
    else remover $f$ de LIST
end

¿Cuál es la complejidad de peor caso de este algoritmo? ¿Cuál es la complejidad para encontrar todas las componentes conexas?

# Camino mínimo

Supuestos

- Grafo dirigido $G = (N, A)$
- costos $c_{ij}$ enterso y no negativos
- Existe un camino dirigido desde el origen $s$ a todos los nodos del grafo.

**Ejemplo** Problema de la mochila.

$$
\begin{array}{cl}
\max & \sum_{i=1}^{n} b_i x_i \\
\text{s.a} & \sum_{i=1}^{n} w_i x_i \leq W \\
 & x_i \in \{0, 1\}
\end{array}
$$

# Algoritmo de Dijkstra

## Idea central en algoritmos de caminos mínimos

- Sea $d(i)$ la etiqueta temporal de distancia mínima desde el origen $s$ al nodo $i$.
- Existe un camino desde $s$ a $i$ de distancia $d(i)$.
- Procedimiento **Update**$(i)$:
  Para todo $j \in A(i)$
  si $d(j) > d(i) + c_{ij}$ entonces $d(j) = d(i) + c_{ij}$ y $pred(j) = i$.

- Sea $d^*(i)$ la distancia mínima desde $s$ a $i$. Si se corre Update$(i)$ cuando $d(i) = d^*(i)$ entonces no se debe ejecutar Update$(i)$ nuevamente.

# Algoritmo de Dijkstra

$S$: Conjunto de nodos etiquetados permanentemente. $\quad d(j) = d^*(j)$ para $j \in S$.

$T$: Conjunto de nodos etiquetados temporalmente. $\quad d(j) \geq d^*(j)$ para $j \in T$.

Set $S = \{s\}$, $T = N \setminus \{s\}$
Set $d(s) = 0$ y $\mathrm{pred}(s) = 0$
Set $d(j) = \infty$ para $j \in T$
Update($s$)
while $S \neq N$ do
    escoja $i \in T$ tal que $d(i) = \min\{d(j) \mid j \in T\}$
    $S = S \cup \{i\}$ y $T = T \setminus \{i\}$
    Update($i$)
end

# Discusión y Complejidad

¿Qué se mantiene en cada iteración? (Invariantes del algoritmo)

¿Qué cosas cambian en cada iteración?

Para demostrar las invariantes son ciertas, use inducción

1) Verifique que invariantes son ciertas despues de inicialización
2) Use inducción. Suponga ciertos en un momento del algoritmo, pruebe que siguen ciertas despues de la siguiente iteración.

   (**iteración:** encontrar nodo min en $T$ y luego un Update.)

# Complejidad Camino Mínimo

Trabajo en un Update($i$):

Trabajo en "encontrar nodo min en $T$":

Complejidad Dijkstra: ¿Se puede mejorar esto? Cuando $G$ no es denso!

- Dijkstra $(c_{ij} \geq 0)$ : $O(n^2)$
- Dijkstra con Radix Heaps $(c_{ij} \geq 0)$ : $O(n + m\log(L))$
- Label correcting sin ciclos negativos : $O(nm)$ Bellman-Ford
- Label correcting con ciclos negativos : $O(nm)$

# Introducción a classes de Complejidad

Un **recognition problem:** es un problema matematico para el cual todas las instancias tienen una respuesta SI o NO. Por ejemplo:

- Para $G = (N, A)$, el tamaño del MST $\leq 7$?
- Para $G = (N, A)$, el valor del max-cut $\geq 15.6$?

¿Qué problemas tienen la posibilidad que tengan un algoritmo eficiente?
Problems that are easy to check?!

A recognition problem $P$ is in $\mathcal{P}$ if some polynomial time algorithm solves $P$. For example:

# Clase $\mathcal{NP}$

Suponga que no se conoce un algoritmo polinomial para algún problema? ¿Qué es lo mejor que se puede decir?

**Class** $\mathcal{NP}$ (NP significa Non-deterministic Polynomial time algorithm) Un prolbema de reconocimiento está en $\mathcal{NP}$ si toda instancia SI tiene un certificado (información adicional) que un algoritmo puede usar para verificar que es una instancia SI en tiempo polinomial.

**Ejemplo**

- Una instancia SI para un problema max-cut $\geq 15.06$ es un grafo para el cual hay un corte $S$ cuya capacidad de los arcos en $[S, \overline{S}]$ es $\geq 15.06$
- El certificado es el corte $S$.
- El algoritmo que chequea el certificado debe ver que $S$ es un subconjunto estricto de $N$, y que la suma de las capacidades de los arcos de $S$ a $\overline{S}$ es $\geq 15.06$

Que pasa a este algoritmo si tenemos una instancia NO del problema de max-cut?

# Reducción polinomial

La idea de poder transformar un problema en otro en tiempo polinomial es central en la clasificación de la complejidad de problemas.

Un problema $P_1$ se reduce polinomialmente a un problema $P_2$ si existe un algoritmo que resuelve $P_1$ resolviendo $P_2$ a lo más un número polinomial de veces.
**Proposición** Si $P_1$ se reduce polinomialmente a $P_2$ y algún algoritmo de tiempo polinomial (in el tamaño de $P_1$) resuelve $P_2$, entonces un algoritmo de tiempo polinomial resuelve $P_1$.

El ciclo Hamiltoniano se reduces al TSP

# $\mathcal{NP}$-complete

En el caso de optimización se dice ($\mathcal{NP}$-hard)

$P \in \mathcal{NP}-\text{complete}$ if
(1) $P \in \mathcal{NP}$
(2) cualquier problema en $\mathcal{NP}$ reduce polinomialmente a $P$.

**Ejercicio:** Sabemos que el problema de ciclo Hamiltoniano es $\mathcal{NP}$-complete. Muestre que TSP es $\mathcal{NP}$-complete.

# Información y ejercicios

Dos sitios interesantes

- https://www.csc.kth.se/tcs/compendium/
- https://www.claymath.org/millenium/p-vs-np/

Reducción de Knapsack a camíno mínimo

Reducción de Stones a Camínos mínimos con restricciones.

# Listado de problemas (fáciles/difíciles)

Los problemas abajo son clásicos en la literatura. Consideraremos un grafo $G = (N, A)$.

1. **Minimum cost spanning tree problem**. (fácil)

   **Minimum k-spanning tree (arbol expansor de peso mínimo con al menos $k$ nodos)** (difícil)

2. **Clique:** Subconjunto $N_1 \subset N$ es un clique si cualquier par de nodos en $N_1$ están conectados por un arco.
   **Maximum clique problem.** (difícil)

# Listado de problemas (fáciles/difíciles)

1. **Independent set:** Subconjunto $N_1 \subset N$ tal que ningún par de nodos en $N_1$ son adyacentes.
   **Maximum independent set problem** (difícil))

   El grafo complemento se define $G^c = (N, A^c)$, donde $(i,j) \in A^c$ ssi $(i,j) \notin A$.
   **Proposición** Si $N'$ es un clique de $G$, entonces $N'$ es un independent set de $G^c$

2. **Chromatic number of a graph:** El número mínimo de colores necesarios para pintar los nodos de un grafo tal que ningun nodo adyacente tenga el mismo color.
   $k$-**coloring problem** (difícil)
   Note que esto es un problema de decisión, existe también la versión de minimización para encontrar el chromatic number.

# Listado de problemas (fáciles/difíciles)

**Proposición** max clique $\leq$ chromatic number.

1. **Minimum/maximum cut problem.** Con respecto a pesos o capacidades. (minimo = fácil/ máximo = difícil)

2. **Cover of a graph:** Un node cover es un subconjunto $N_1 \subset N$ tal que todo arco en $A$ es adyacente a al menos un node en $N_1$.
   **Minimum weight cover.**(difícil)

3. **Assignments and matchings:** Un matching (emparejamiento) es un subconjunto de arcos $A_1 \subset A$, tal que todos los nodos son incidentes a lo mas a un arco en $A_1$. Un assignment es un matching bipartito (i.e. un matching en un grafo bipartito).
   **Maximum/minimum matching** (fácil)

# Listado de problemas (fáciles/difíciles)

**Arc routing/sequencing problems**

1. **Euler tour:** Un tour de Euler es un camino que empieza en un nodo dado, visita todos los arcos exactamente una vez y vuelve al nodo inicial.
   **Proposición** $G$ contiene un tour de Euler ssi $G$ tiene grado par
   **Euler tour problem** Dado un grafo, ¿tiene un tour de Euler? Versión de optimización, tour de Euler de costo mínimo. (fácil)

2. **Chinese postman tour** Un tour (paseo direccionado cerrado) que visita cada arco al menos una vez
   **Chinese postman problem** El grafo $G$ tiene un tour de cartero Chino? (fácil si el grafo es dirigido o no es dirigido / difícil si algunos arcos dirigidos y otros no)

# Listado de problemas (fáciles/difíciles)

**Vertex routing/sequencing problems**

1. **Hamiltonian cycle:** Un ciclo dirigido que visita cada nodo en el grafo exactamente una vez.
   **Hamiltonian cycle problem** ¿El grafo $G$ tiene un ciclo Hamiltoniano? (difícil)

   - Hamiltonian path problem. (difícil)
   - **TSP:** (Traveling Salesman Problem) Ciclo Hamiltoniano de menor costo. (difícil)
   - **TSP-recognition:** ¿Existe un tour Hamiltoniano de costo $k$? (difícil)

# Listado de problemas (fáciles/difíciles)

1. **Satisfiability problem:** (2-SAT, 3-SAT, ...) ¿Se pueden satisfacer ciertas expresiones lógicas fijando un conjunto de valores 0/1?

   Para $n$ literales $x_i$, que pueden ser 0 o 1.
   $k$-SAT, está hecho de clausulas "o" con $k$ literales: $C_j = x_3 \vee x_5 \vee x_{18}$.
   Las clausulas se unen con "y" lógicos: $C_1 \wedge C_2 \wedge \cdots \wedge C_m$.

   Es el primer problema demostrado ser NP-completo [Cook 1971]