Técnicas Algorítmicas

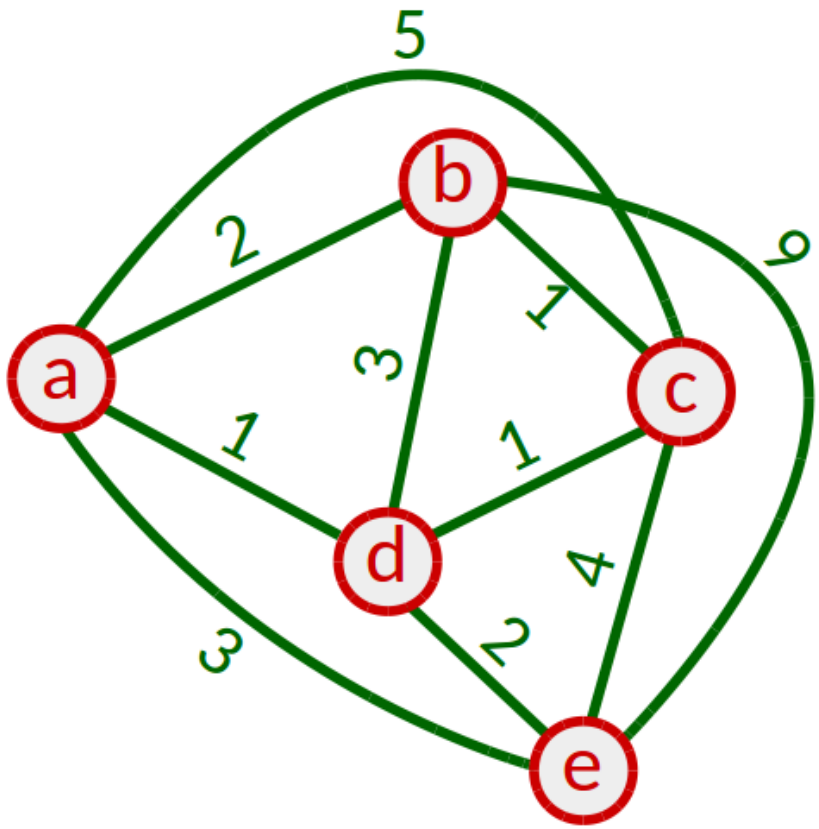
Prof. Agustín Gravano

MiM - UTDT - Segundo semestre de 2020

Clase 5: Heurísticas

Problema del Viajante de Comercio (TSP)

Dados un conjunto de ciudades y la distancia entre cada par de ellas, hallar el recorrido más corto que visita cada ciudad exactamente una vez y retorna a la ciudad de origen.



Problema del Viajante de Comercio (TSP)

Búsqueda exhaustiva con backtracking.

Función Viajante de Comercio:

- Si todavía quedan ciudades por visitar:
  - Para cada ciudad c que falta visitar:
    - Avanzar un paso: visitar c en este momento.
    - Recursivamente, obtener el mejor recorrido r_c desde acá.
    - Retroceder un paso: no visitar c en este momento.
  - De todos los recorridos r_c vistos, devolver el mejor.
- Si ya no quedan ciudades por visitar:
  - Terminamos de construir un recorrido. Devolverlo.

Este algoritmo demanda $O(2^n)$ operaciones (n: cantidad de ciudades en el mapa). Ver `analisis_temporal.py`.

Problemas *difíciles*

Problemas clásicos de optimización combinatoria:

- Viajante de comercio: Recorrer todas las ciudades minimizando la distancia recorrida.
- Problema de la mochila: Cargar una mochila con la combinación más valiosa de ítems.
- Corte de materiales: Recortar piezas de un material minimizando las pérdidas.
- Asignación de recursos: Asignar recursos maximizando la cantidad de pedidos satisfechos.

No se conocen soluciones eficientes (polinomiales).

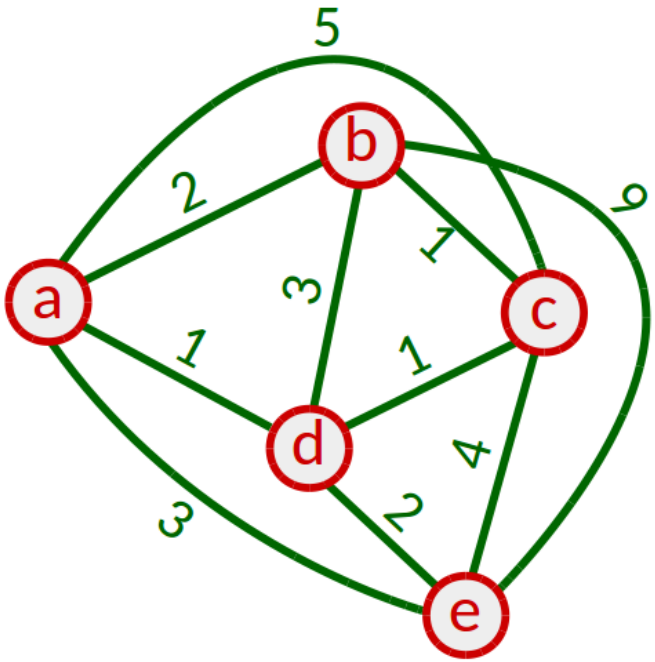
Backtracking nos permite resolver estos problemas en forma exacta, pero demanda $O(2^n)$.

Alternativa: Heurísticas de programación.

- Son algoritmos **aproximados**, pero eficientes.

Técnica greedy (ávido, voraz, goloso)

- Al construir una solución, tomar en cada paso la mejor decisión local.
- Viajante de comercio:



En cada paso, ir a la ciudad más cercana que no hayamos visitado aún.

1. a → d
2. d → c
3. c → b
4. b → e

Solución greedy: [adcbea]

Distancia recorrida: 15

Técnica greedy (ávido, voraz, goloso)

Es fácil encontrar ejemplos en los que un algoritmo greedy falla miserablemente para TSP.

Técnica greedy (ávido, voraz, goloso)

Otro ejemplo de aplicación: problema de cambio en monedas.

Tenemos suficientes (infinitas) monedas de denominaciones $\{1, 5, 10, 25, 50\}$. Queremos pagar un monto M con una cantidad mínima de monedas.

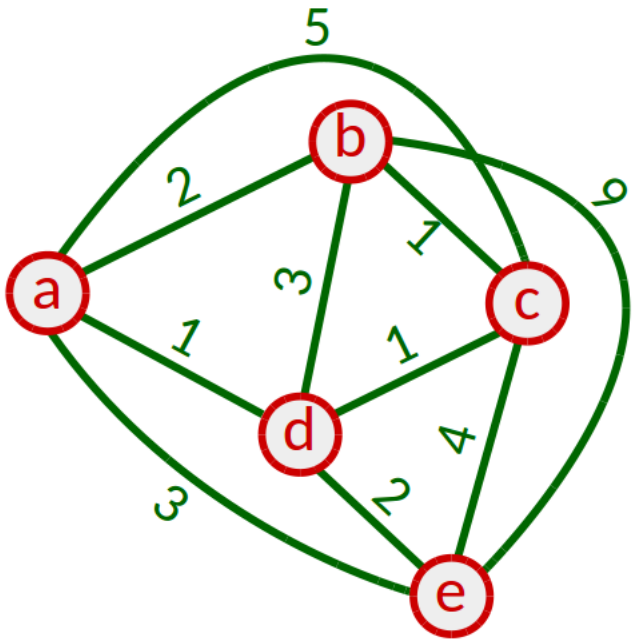
Por ejemplo, si $M = 30$, hay varias combinaciones posibles:

- 30×1 (30 monedas)
- 3×10 (3 monedas)
- 6×5 (6 monedas)
- $1 \times 25 + 5 \times 1$ (6 monedas)
- $1 \times 25 + 1 \times 5$ (2 monedas)
- ...

Algoritmo greedy: Elegir en cada paso la moneda de máxima denominación, hasta llegar a M. Este algoritmo encuentra la solución óptima para monedas con esas denominaciones.Pero puede fallar para otras denominaciones (ej: $\{1, 3, 4\}$, $M = 6$).

Xuan Cai (2008), "Canonical Coin Systems for Change-Making Problems" https://arxiv.org/abs/0809.0400

Problema del Viajante de Comercio (TSP)



Supongamos que tenemos una solución S, construida de alguna manera (por ejemplo, al azar).

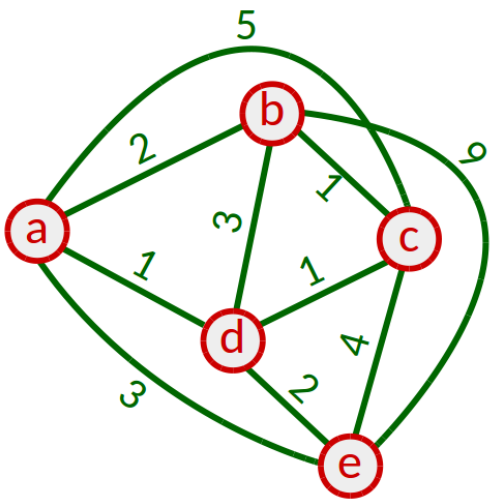
Definimos soluciones vecinas de S mediante modificaciones simples a S. Por ejemplo: intercambiar dos ciudades.

Ejemplo: S = [adcbea]

Vecinos de S: [acdbea], [abcdea], [aecbda], etc.

Técnica de búsqueda local

- Comenzar con una solución S elegida de algún modo (p.ej., al azar, o bien con un algoritmo greedy).
  1. Explorar los *"vecinos"* de S
  2. $S \leftarrow$ mejor vecino encontrado
  3. Repetir hasta no poder mejorar
- Viajante de comercio:



Vecino = intercambiar dos ciudades cuales- quiera del recorrido (excepto 1ra y última)

1. [acedba] (solución random, dist=16)
2. [adecba] (mejor vecino: c $\leftrightarrow$ d, dist=10)
3. [aedcba] (mejor vecino: d $\leftrightarrow$ e, dist=9)

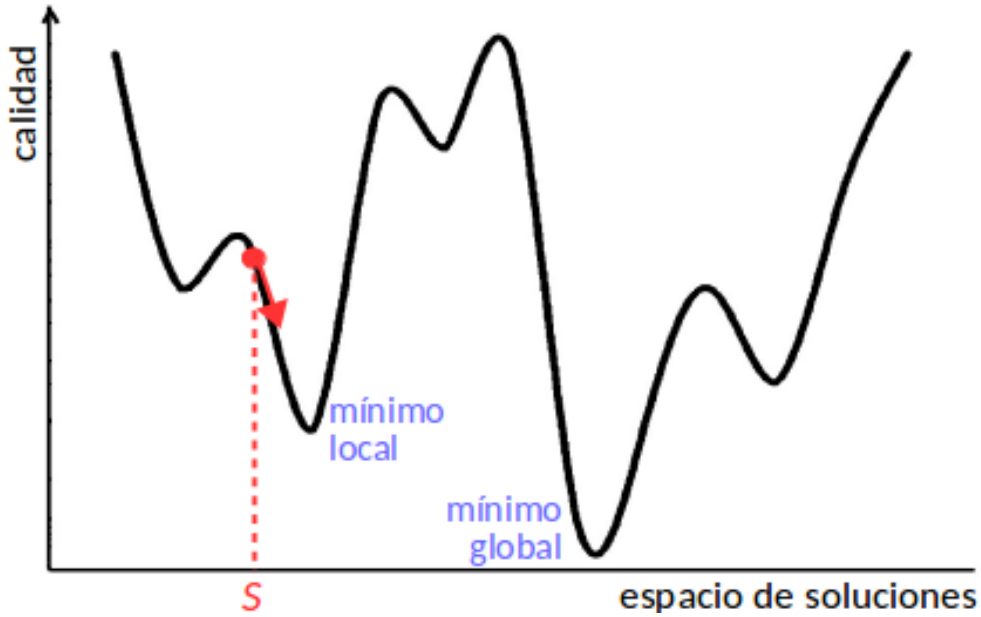
Solución de búsqueda local: [aedcba]

Distancia recorrida: 9

Técnica de búsqueda local iterativa

Problema:
Riesgo de quedar atrapado en un mínimo local

¿Cómo podemos atacar este problema?



1. Repetir hasta cumplir una condición de corte (*):
   a) $S \leftarrow$ Solución aleatoria.
   b) $S \leftarrow$ Búsqueda local a partir de S.
   c) Si S es la mejor solución hallada hasta ahora, guardarla.
2. De todas las soluciones guardadas, devolver la mejor.

(*) Ej: cantidad máxima de iteraciones; tiempo máximo de ejecución; etc.

Técnica GRASP

Greedy Randomized Adaptive Search Procedure:

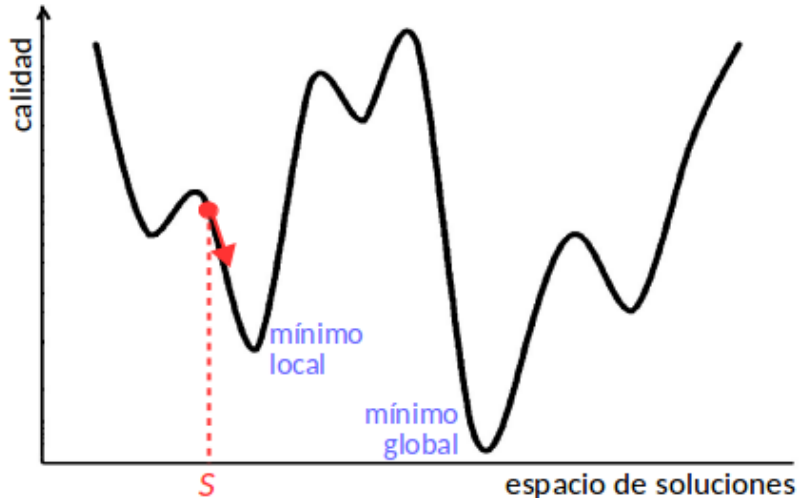
1. Repetir hasta cumplir una condición de corte:
   a) $S \leftarrow$ Solución **greedy aleatoria**.
   b) $S \leftarrow$ Búsqueda local a partir de S.
   c) Si S es la mejor solución hallada hasta ahora, guardarla.
2. De todas las soluciones guardadas, devolver la mejor.

Una solución **greedy aleatoria** se construye de manera similar a greedy, pero en cada paso:

1. Construir una lista L con las k mejores decisiones locales que podríamos tomar
2. Elegir al azar una decisión de L para tomar en el paso actual

Observación: si $k = 1$, equivale a la técnica greedy

Más ideas para escapar de los mínimos locales

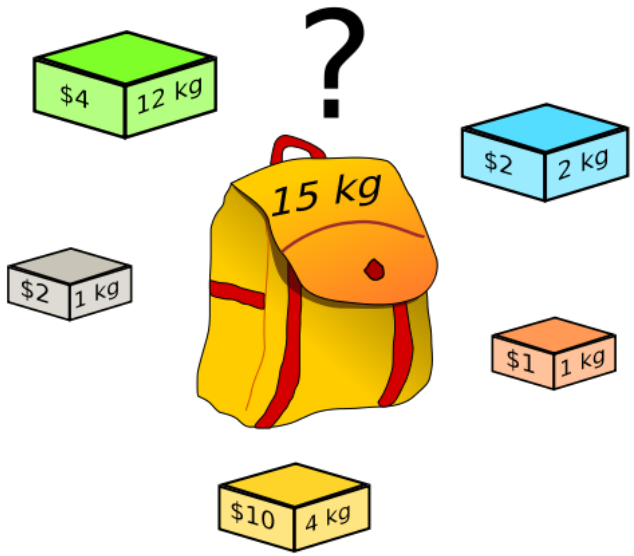


- Tabu search: Se mantiene una lista de soluciones ya visitadas, para evitar repetirlas más adelante, y así explorar nuevos vecindarios.
- Simulated annealing: Con cierta probabilidad T (llamada *temperatura*), se permite pasar a vecinos inválidos, que no cumplen las condiciones para ser soluciones válidas. T empieza alta y va bajando lentamente.

Otro ejemplo: Problema de la mochila

Dados una mochila que puede cargar hasta N kg, y un conjunto de ítems, cada uno con un peso (en kg) y un valor (en \$), encontrar el subconjunto de ítems con máximo valor que quepa en la mochila.

Ejemplo de aplicación: conformación de un portfolio de acciones



Ejercicio: Pensar tres algoritmos con las siguientes técnicas para resolver el problema de la mochila:

- Backtracking (solución exacta)
- Greedy (solución aproximada)
- Búsqueda local (solución aproximada)

Heurísticas de programación

Otras heurísticas notables:

- Programación genética
- Colonias de hormigas *(ant colonies)*
- Enjambre de partículas *(particle swarm optimization)*

Lecturas sugeridas:

- El-Ghazali Talbi, "Metaheuristics: From Design to Implementation", Wiley-Blackwell, 2009. `http://www.lifl.fr/~talbi/metaheuristic/`
- Mauricio G.C. Resende, Celso C. Ribeiro, "Greedy Randomized Adaptive Search Procedures". Sección Extras del campus → archivo `GRASP.pdf`
- Francisco Herrera, "Introducción a los Algoritmos Metaheurísticos". Sección Extras del campus → `TutorialHerrera2006.pdf`.

Repaso de la clase de hoy

- Heurísticas de programación:
  - Greedy
  - Búsqueda local
  - Búsqueda local iterativa
  - GRASP
- Ejemplos:
  - Problema del viajante de comercio
  - Problema de la mochila

Con lo visto hoy, ya pueden resolver la sección 5 de la guía de ejercicios y todo el Trabajo Práctico Final.

¡Fin! :-)