Microsoft 365

Branch and Cut for TSP

Caputo Salvatore & Improta Gaetano

Università degli Studi della Campania *Luigi Vanvitelli*

TSP

Obiettivo: Data una lista di "città" (nodi), trovare il **percorso più breve** possibile che:

- Visiti ogni città **esattamente una volta**.
- Torni alla città di partenza.

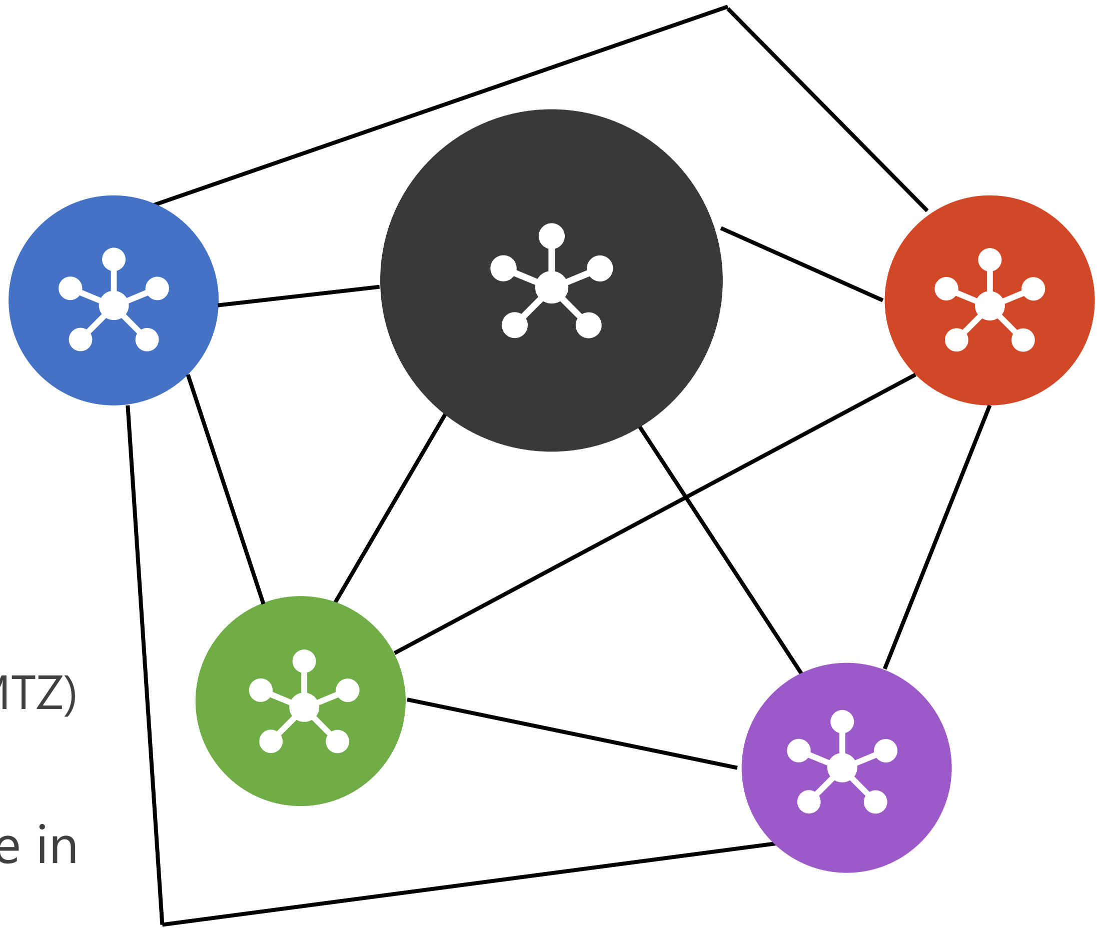
$$\min Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \cdot x_{ij}$$

Vincoli:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \quad , \quad \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j$$

$$u_i - u_j + n \cdot x_{ij} \leq n - 1, \quad \forall i, j \in \{2, \dots, n\}, i \neq j \quad \text{(MTZ)}$$

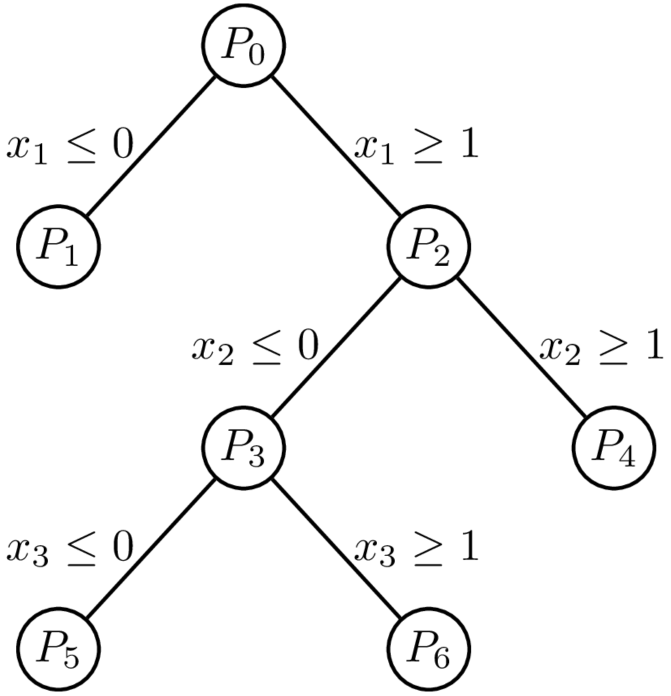
Problematica: Il numero di percorsi possibili cresce in modo **fattoriale** $\frac{(n-1)!}{2}$



Università degli Studi della Campania *Luigi Vanvitelli*

Branch and Bound

Idea di base: Immagina di esplorare l'intero albero di tutti i (n−1)! percorsi possibili. Invece di esplorarli tutti fino alla fine, il BnB "*pota*" interi rami dell'albero non appena capisce che *non porteranno mai* alla soluzione migliore.



1. **Branch:** «Divide». L'algoritmo inizia a costruire l'albero delle decisioni, passo dopo passo (esattamente come nell'esempio della forza bruta).

2. **Bound:** «Sfoltisci», è il cuore dell'algoritmo. È ciò che lo rende intelligente. Per ogni "nodo" (percorso parziale) creato, calcola un **Limite Inferiore** (una **stima ottimistica** del costo *minimo* che un tour completo *potrebbe* avere, *se* includesse quel percorso parziale.)

Università degli Studi della Campania *Luigi Vanvitelli*

Branch and Bound: Problemi

1. Stima del LB

La stima del **Lower Bound** nel BnB è una stima **ottimistica**: i vincoli di interezza sono rilassati.

Se il Limite Inferiore (la stima ottimistica) è quasi sempre migliore della soluzione trovata passo fin’ora, l'algoritmo non può potare quasi nulla.

Conseguenza: la sua complessità temporale nel caso peggiore resta **esponenziale.**

2. Consumo di Memoria

Il B&B non esplora solo un percorso alla volta; deve tenere traccia di *tutti* i percorsi parziali "promettenti" che ha messo in pausa per esplorarne un altro.

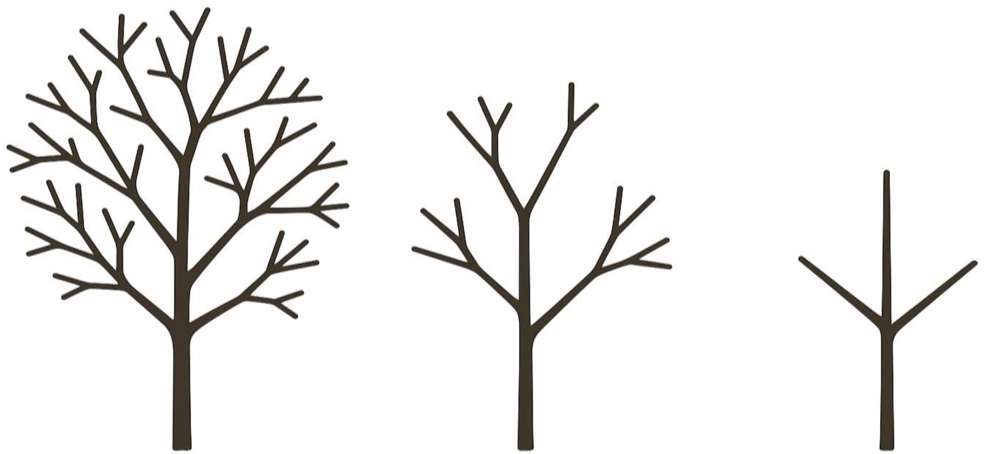
Come? Questi percorsi parziali vengono memorizzati in una struttura dati.

Branch and Cut

L'algoritmo **Branch and Cut** (BnC) è una delle tecniche più efficaci per risolvere problemi di Programmazione Lineare Intera , come il Problema del Commesso Viaggiatore.

Nasce come un potenziamento del metodo classico Branch and Bound, il quale come abbiamo visto soffre spesso di una limitazione critica:

- il **"limite inferiore"** calcolato tramite il Rilassamento Lineare è **troppo debole per permettere un'efficiente potatura** dell'albero di ricerca.



Branch and Cut

L’aspetto cruciale è il ***Cutting***, ovvero viene aggiunta la:

Fase di Cut: l'algoritmo identifica dinamicamente i vincoli del problema che vengono violati dalla soluzione rilassata (i sottocicli nel TSP) e li aggiunge iterativamente al modello.

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset V, 2 \leq |S| \leq n - 1$$

con V insieme di tutti i nodi (città) ed S un suo sottoinsieme non banale.

Algoritmo: algoritmo_BnC

Università degli Studi della Campania *Luigi Vanvitelli*

Branch and Cut

Esempio: 4 città (A, B, C, D)

supponiamo che la soluzione più economica trovata sia questa:

- Un percorso che va da **A a B e torna subito ad A**.
- Un percorso che va da **C a D e torna subito a C**.

La soluzione è formalmente intera e minimizza il costo, ma **non è un tour valido** per il Commesso Viaggiatore, ci sono due sottocicli!

➢ Entra in gioco il **Cut:**

Vincolo aggiunto: la somma di tutti i percorsi che vanno da A a B, e da B ad A, deve essere al massimo 1. (soluzione precedente non più valida)

Si forza una nuova soluzione, e quindi un LB più costoso

➢ implicitamente verranno scartate tutte le soluzioni con questo sottociclo, senza dover ramificare inutilmente.

Branch and Cut

Branching:

Quando risolviamo il modello rilassato per ottenere il nostro limite inferiore, il risolutore spesso ci dà risposte **frazionarie** (es. x_AB = 0.6).

Questo valore è ottimo per la stima, ma non è una decisione reale: non possiamo percorrere un arco al 60%!

Il **Branching** è il meccanismo che utilizziamo per correggere questo errore di stima e **forzare** la variabile frazionaria a diventare una **decisione binaria (0 o 1)**.

Vengono creati due nuovi rami dell'albero (problemi):

- Ramo 1: vincolo x_AB = 0 (non percorriamo l'arco x_AB)
- Ramo 2: vincolo x_AB = 1 (percorriamo l'arco x_AB)

In questo modo, **tutte le soluzioni intere possibili** verranno esplorate.

Università degli Studi della Campania *Luigi Vanvitelli*

Grazie dell’attenzione

Università
degli Studi
della Campania
Luigi Vanvitelli