Algoritmo Branch and Cut

L'algoritmo **Branch and Cut (BnC)** è una delle tecniche più efficaci per risolvere problemi di Programmazione Lineare Intera (**PLI**), come il Problema del Commesso Viaggiatore. Nasce come un potenziamento del metodo classico **Branch and Bound (BnB)**, il quale soffre spesso di una limitazione critica: il "limite inferiore" (**Lower Bound**) calcolato tramite il Rilassamento Lineare (PL) è troppo debole per permettere un'efficiente potatura (pruning) dell'albero di ricerca, essendo costretto a ignorare un numero esponenziale di vincoli per essere veloce.

Il Branch and Cut risolve questo problema aggiungendo una fase di "**Cutting**" (generazione di piani di taglio). L'algoritmo identifica **dinamicamente** i vincoli del problema che vengono violati dalla soluzione rilassata (ad esempio, i sottocicli nel TSP) e li aggiunge iterativamente al modello.

Questo processo rafforza la formulazione, produce un limite inferiore molto più stretto e accelera drasticamente la convergenza verso la soluzione ottima globale. Questo documento ne analizza i passaggi fondamentali.

Fase 1*: *Inizializzazione e Rilassamento

1. **Inizializzazione**: Si crea l'albero di ricerca con il problema originale come nodo radice. Si stabilisce un Limite Superiore (UB): si esegue un'euristica veloce per trovare una prima soluzione intera valida il cui costo è l' UB .

2. **Rilassamento PL**: Si seleziona un nodo k da esplorare. Si risolve il Rilassamento Lineare (PL) standard di quel nodo, ignorando i vincoli di integrità e, inizialmente, i vincoli espliciti non inclusi nella formulazione di base (come i Vincoli di Eliminazione dei Sottocicli, SEC, nel TSP).

3. **Calcolo del LB**: Il costo della soluzione del PL, Z_k^*, diventa il Limite Inferiore (LB) per il nodo k. La soluzione ottenuta è x^*.

Fase 2: Potatura e Generazione di Cut

Inizia la fase di verifica della soluzione e di rafforzamento del modello:

4. **Verifica di Potatura (Bounding)**: Si confronta il LB con l'UB.
   - Se $Z_k^* \geq$ UB: Il nodo viene scartato (potato). Si va al Passo 8.

5. **Verifica della Validità del Cut (Separation)**: Se il nodo non è potato, si esamina la soluzione x^* per vedere se viola eventuali vincoli che, pur essendo validi per le soluzioni intere, non sono inclusi nella formulazione attuale.
   - Generazione di Cut: Si lancia un algoritmo di separazione.

6. **Aggiunta dei cut (Rafforzamento)**:
   - Se viene trovato un vincolo violato (Cut): Il vincolo viene aggiunto al set di vincoli del problema PL del nodo k.
   - Vantaggio: Questo vincolo taglia via la soluzione x^* e restringe il poliedro P. Si torna al Passo 2 (Risoluzione del PL) per calcolare un nuovo, più forte LB. Si ripete questo ciclo di separazione e risoluzione finché non si trovano più cuts significativi.

Fase 3: Branching e Aggiornamento

Una volta che il LB è stato rafforzato al massimo con i cuts:

7. **Verifica di Integrità**: Si esamina la soluzione x^* per vedere se è intera.
   - Se x^*è intera: La soluzione è valida. Si aggiorna l'UB con Z_k^* (se Z_k^*< UB). Il nodo viene scartato.
   - Se x^* è frazionaria: Si esegue il Branching. Si sceglie una variabile non intera x_i e si creano due nuovi nodi: uno con $x_i = \lfloor x_i * \rfloor$e uno con $x_i = \lceil x_i * \rceil$. Si aggiungono i nuovi nodi all'elenco di quelli da esplorare.

8. **Terminazione**: Si passa al nodo successivo non esplorato. L'algoritmo termina quando non ci sono più nodi da esplorare (o potare). L'ultimo UB è la soluzione ottima globale provata.