MIP Practice

MIP Solvers Implementation

Domenico Salvagnin

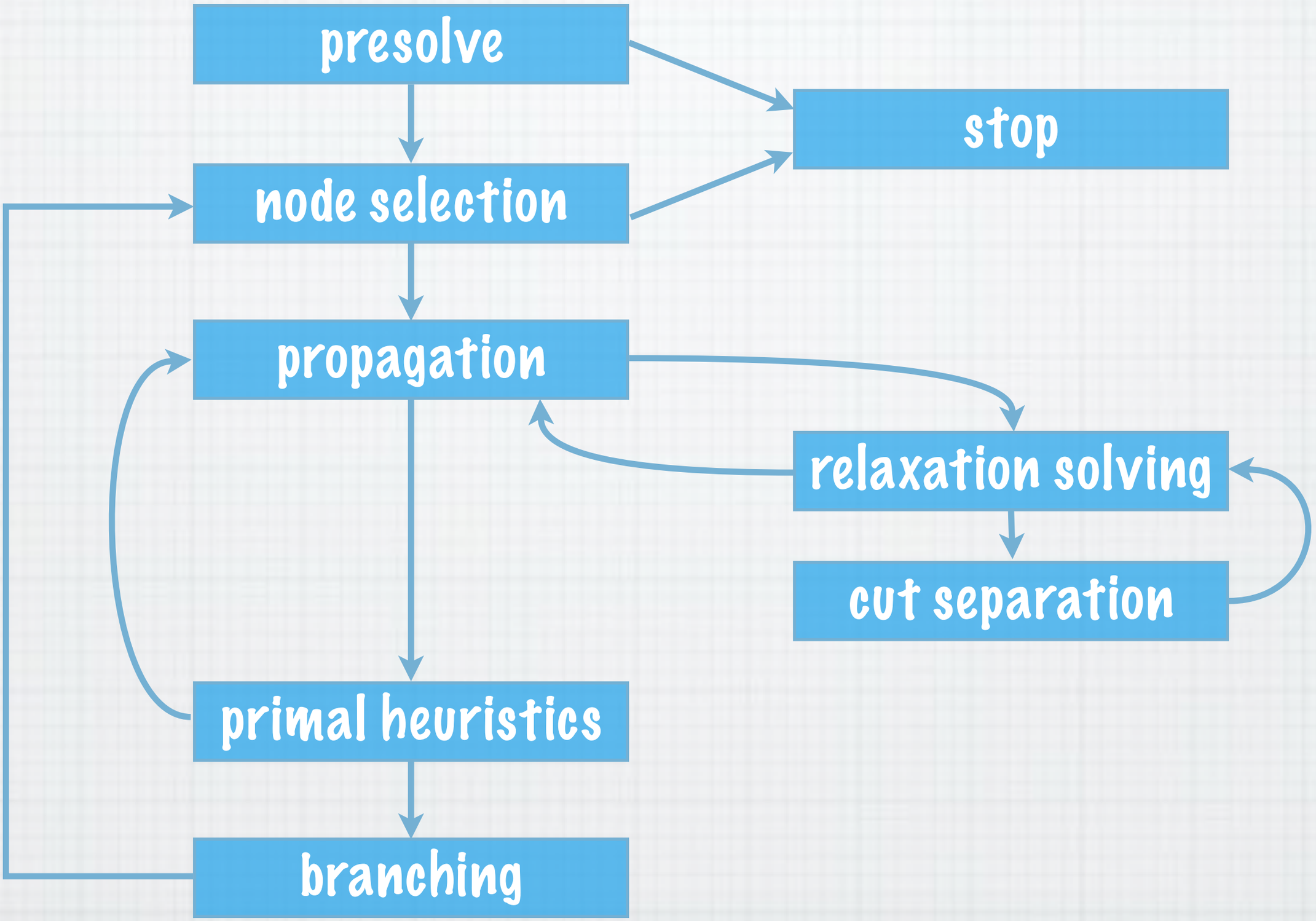
Branch-and-bound

* Branch-and-bound ha sempre due bound a disposizione:
  * **bound primale**: valore della migliore soluzione feasible corrente (incumbent)
  * **bound duale**: valore del miglior rilassamento dei nodi non ancora processati
* La differenza tra i due si dice **optimality gap**
* Tali bound convergono durante l'esecuzione dell'algoritmo (alla fine gap = 0)
* L'efficienza dell'algoritmo dipende dalla velocità con cui convergono!

Software Branch-and-cut

* In pratica i software che implementano un algoritmo branch-and-cut fanno ricorso a molte tecniche per aumentare l'efficienza del metodo:
  * implementazioni efficienti del simplesso
  * strategie di branching e esplorazione (node selection) raffinate
  * preprocessing (formulazioni equivalenti + forti)
  * primal heuristics (per trovare prima soluzioni intere)
  * constraint propagation (deduzioni)
  * strategie raffinate di separazione (e selezione) dei tagli

Branch-and-Cut Sketch



Presolve

* Scopi del presolve:
  * ridurre dimensioni problema
  * migliorarne proprietà numeriche
  * dimostrare infeasibility/unboundedness senza risolverlo
* Tre categorie principali di presolve:
  * riduzioni primali (feasibility reasonings)
  * riduzioni duali (optimality reasonings)
  * scaling

Presolve: tecniche base

1. empty row (redundant o infeasible)
2. empty column (fissaggio ad un bound o unboundedness)
3. variabile infeasible
4. variabile fissata
5. singleton row (fissaggio variabile o infeasible)
6. free column singleton (elimino una riga e una colonna)

N.B. Ciascuna di queste riduzioni può scatenarne altre, quindi generalmente si eseguono in loop fino a che non si raggiunge uno stato stazionario (fixpoint)

Presolve: Bound Strengthening

Dato un vincolo lineare con coefficienti positivi:

$$LB \leq \sum a_j x_j \leq UB$$

e variabili bounded $[l_j, u_j]$, possiamo calcolare le seguenti attività minime e massime:

$$L_{\min} = \sum a_j l_j \qquad L_{\max} = \sum a_j u_j$$

e aggiornare i vincoli sulle variabili:

$$x_j \leq l_j + \frac{UB - L_{\min}}{a_j} \qquad x_j \geq u_j + \frac{LB - L_{\max}}{a_j}$$

(tali valori possono essere opportunamente arrotondati per variabili intere e il tutto si può estendere a vincoli con coefficienti misti e a variabili unbounded)

Presolve: miscellanea

* Il calcolo delle attività minime e massime si può eseguire efficientemente ed in modo incrementale
* Ulteriori tecniche di presolve:
  * righe/colonne duplicate
  * aggregazione
* Necessario tenere traccia delle operazioni di presolve per ricostruire soluzione ottima problema originario (presolve stack)
* Alcune riduzioni possono essere incompatibili con eventuali modifiche al problema: attenzione!

Node selection

- Algoritmo mantiene una coda Q di nodi non processati
- Grado di libertà nella scelta di quale nodo estrarre da Q
- Due strategie principali:
  - Depth First
  - Best Bound First
- Le due strategie hanno caratteristiche complementari, quindi spesso si ricorre a strategie ibride

Branching

* Data una soluzione frazionaria, in generale ho molte possibili variabili candidate per il branching
* Fattore molto importante per l'efficacia dell'algoritmo, ma non è nota nessuna strategia che minimizzi con certezza il numero dei nodi dell'albero
* In pratica vengono usate le seguenti:
  * most fractional variable (semplice ma poco performante)
  * strong branching (attualmente la migliore, ma computazionalmente molto pesante)
  * pseudocost branching (stima dello strong branching)

Strong Branching

- Calcola la variazione del bound duale in entrambi i nodi generati dal branching su ciascuna variabile frazionaria
- Sceglie (in modo greedy) la variabile che porta alla maggiore variazione del migliore dei bound
- Costo molto alto: richiede due riottimizzazioni duali per ogni variabile candidata!
- Tecniche di velocizzazione:
  - scelta preliminare di un sottoinsieme di candidati
  - limite di iterazioni di simplesso

Pseudocost Branching

* Stima la variazione del bound duale indotto dal branching su una variabile basandosi sulla storia dei branching precedenti
* Versione "euristica" dello strong branching, senza risolvere lineari
* Molto più veloce dello strong branching, ma meno efficace
* All'inizio, proprio quando le decisioni sono più importanti, la quantità di informazioni raccolte è scarsa, e quindi il metodo è poco affidabile nel momento più critico!

Domain Propagation

* Consiste nel rafforzamento dei domini delle variabili a partire dai vincoli originari del problema e dai vincoli di branching al nodo corrente
* Permette di rafforzare rilassamento lineare del singolo nodo

$$\begin{cases} 2x_1 + 3x_2 \leq 4 \\ x_1 = 1 \end{cases} \implies x_2 = 0$$

* Forma ristretta di presolve
* Importante che l'implementazione sia efficiente ed incrementale
* Reduced Cost Fixing (propagazione duale)

Cut Separation

- Molte classi di tagli studiate in letteratura:
  - knapsack cover cuts
  - flow cover cuts
  - mixed integer rounding cuts (MIR)
  - Gomory mixed integer cuts (GMI)
  - Chvatal-Gomory cuts (CG)
  - implied bound cuts
  - ...

Cut Separation (2)

* Poichè la convergenza dell'algoritmo è garantita dal branching, le procedure di separazione possono essere euristiche
* Viene preferita la generazione di tagli globalmente validi da gestire in una struttura dati comune a tutti i nodi (pool), ma sono supportati anche i tagli locali
* Molti parametri:
  * quanti tagli generare? (round di tagli)
  * quando? (solo al nodo radice, ad ogni nodo, ogni tanto...)

Primal Heuristics

- Una soluzione ammissibile è necessaria per:
  - bounding
  - reduced cost fixing
  - altre improving heuristics (RINS, LB, crossover)
- L' algoritmo classico aspetterebbe di trovare una soluzione intera del rilassamento...troppo poco probabile!
- Si ricorre quindi ad euristiche primali:
  - rounding
  - diving
  - feasibility pump

Feasibility Pump: schema di base

Genera due traiettorie di punti che soddisfano i vincoli del problema in modo complementare

Come si ottiene un punto intero da uno frazionario?

Arrotonda le componenti intere e lascia invariate le altre

$$\tilde{x}_j = \begin{cases} [x_j^*] & j \in J \\ x_j^* & j \notin J \end{cases}$$

Come si ottiene un punto frazionario da uno intero?

Trova il punto in P più vicino in norma L_1

$$x^* = \arg\min\{\Delta(x, \tilde{x}) : x \in P\}$$

Qualche link...

* ZIB SCIP: http://scip.zib.de
* Coin-OR CBC: http://www.coin-or.org/projects/Cbc.xml
* T. Achterberg "Constraint Integer Programming", PhD thesis, Technische Universität Berlin, 2007