

Parallel Branch & Bound

Bernard Gendron
Université de Montréal
gendron@iro.umontreal.ca



Outline

- Mixed integer programming (MIP) and branch & bound (B&B)
 - Linear programming (LP) based B&B
 - Relaxation and decomposition
 - Sequential search strategies
- Parallel B&B
- Some representative examples
- Lessons from past experience
- Software tools
- Research issues



Integer Programming (IP)

- Combinatorial optimization (CO) problems can often be formulated as MIP models:

$$\min f(x, y)$$

$$Ax + By \geq b$$

$$Dx + Ey \geq e$$

$$x, y \geq 0$$

$$y \text{ integer}$$

- When facing NP-hard problems, we often use a B&B algorithm where the lower bounds are computed using some convex approximation of the MIP model



LP-based B&B

- The most common approximation is obtained by relaxing the integrality constraints: if the objective function is linear, we obtain the LP relaxation
- At each node of the B&B tree, the lower bound is obtained by solving the LP relaxation
- Upper bounds are obtained when all y variables have integral values in an LP optimal solution
- Branching: one y variable with a non-integral value y^* is selected and we create two subproblems

$$y \leq \lfloor y^* \rfloor \text{ and } y \geq \lceil y^* \rceil$$



Mathematical Decomposition

- For some models, all the constraints/variables cannot (in practice) be enumerated a priori
- Decomposition methods are then used:
 - Row generation or *cutting-plane* methods
 - *Column generation* or Dantzig-Wolfe methods
- At each iteration, these methods solve a smaller model defined over a subset of constraints/variables
- A subproblem is then solved to identify:
 - Constraints violated by the current solution (*separation*)
 - Variables that may be added to improve the value (*pricing*)
- The methods stop when no more constraints/variables can be generated



Decomposition in B&B

- Decomposition methods are often used in B&B to compute lower bounds *at every node of the tree*:
 - Row generation + B&B = Branch & Cut (B&C)
 - Column generation + B&B = Branch & Price (B&P)
- A key issue is then how to share the constraints/variables among the nodes of the tree in order to avoid generating them over and over
- In sequential implementations, pools of constraints/variables are used
- How to implement such pools in parallel environments is a major issue



Lagrangian Relaxation

- Lagrangian relaxation is used when MIP models of CO problems reveal specialized subproblems
- In our model, suppose that the problem without the first set of constraints is “easy” to solve
- Relaxing these constraints, introducing them in the objective with multipliers $\lambda \geq 0$, we obtain a lower bound:

$$\min f(x, y) + \lambda(b - Ax - By)$$

$$Dx + Ey \geq e$$

$$x, y \geq 0$$

$$y \text{ integer}$$



Lagrangian Relaxation in B&B

- There is a huge literature on methods for finding the best values for the multipliers
- Linear objective: the best Lagrangian bound improves upon (or is equal to) the LP bound
- Even when the Lagrangian and LP bounds are equal, Lagrangian relaxation can provide more efficient bounding methods than LP relaxation
- In a Lagrangian-based B&B, branching often relies on the multipliers
- Reoptimization is an issue (very efficient in LP)



Preprocessing

- Methods used to fix variables, eliminate redundant constraints,... without changing the optimal solution
- Can be repeated at every node of the B&B tree
- *Probing*: looks at the implication of having $y = \delta$; if the problem becomes infeasible then $y \neq \delta$
- The *reduced cost* of variable y is a measure of the increase in the lower bound when we change the value of y
- *Reduced cost fixing*: if the problem with $y = \delta$; cannot be optimal ($Z' + C_y \geq Z^u$) then $y \neq \delta$



Sequential Search Strategies: Best-First

- Selects the active node in the list with the smallest lower bound
- Often implemented using *eager* evaluation: at node creation, bounds are immediately evaluated
- Advantage: among all selection strategies, minimizes the number of generated nodes
- But only if bounding and branching do not depend on when they are performed!
- Disadvantage: lot of memory (expands the tree in all directions)



Sequential Search Strategies: Depth-First

- Selects the active node in the list which is the deepest in the tree
- Often implemented using *lazy* evaluation: bounds are evaluated only when the node is selected
- Advantages:
 - Minimizes memory requirements
 - Helps reoptimization
 - Quickly finds feasible solutions
- Disadvantage: a lot of generated nodes, if the upper bounds are not good enough



Parallel B&B

- Classification from Gendron and Crainic (1994)
- Type 1: parallelism in bounding and branching
- Type 2: parallel search of the tree
- Type 3: concurrent explorations of several trees
- Within Type 2 (the most common), we distinguish:
 - Synchronous (S) or Asynchronous (A)
 - Single (SP) or Multiple Pool (MP)
- In MP algorithms:
 - Collegial
 - Grouped
 - Mixed



Design Issues in Parallel B&B

- Shared memory or message passing
- Synchronous or asynchronous
- Managing the list of nodes: single or multiple pools
- Initial node generation and allocation
- Node allocation and sharing: *dynamic load balancing*
 - On request
 - Without request
 - Combined
- Managing the incumbent
- Termination detection



ASP Example: Concurrent Heap

- Nageshwara Rao, Kumar 1988
- Each processor picks the next node to examine in a central pool
- The pool is managed as a concurrent heap data structure: several processors may access it simultaneously
- Much more efficient than a sequential heap
- But contention of access still a problem



ASP Example: 0-1 MIP

- Bixby, Cook, Cox, Lee 1999
- Integrates most features of modern MIP solvers: preprocessing, reduced-cost fixing, cutting-plane (at the root only), heuristics
- Implemented with TreadMarks: a shared-memory parallel programming environment able to run on distributed-memory machines
- Initialization: run the sequential algorithm until enough ($> p$) nodes are generated



AMP Example: Mixed Organization

- Kumar, Ramesh, Nageshwara Rao 1988
- Each processor has its own local pool, but there is also a global pool, called *blackboard*
- Each processor picks the best node in its local pool and compares it to the best node in the blackboard
 - If it is much better, the processor sends some of its good nodes to the blackboard
 - If it is much worse, the processor transfers some good nodes from the blackboard to its local pool
 - If it is comparable, the processor branch on its best node
- Dynamic load balancing without request



AMP Example: Load Balancing

- Quinn 1988
- Collegial organization on a hypercube
- Dynamic load balancing without request: at each iteration, each processor sends one subproblem to one of its neighbors
- Four criteria to decide which subproblem to send:
 - Any one of the newly generated nodes
 - The newly generated node with smallest lower bound
 - The second best node among all nodes in the local pool
 - The best node among all nodes in the local pool
- Third and fourth strategies perform best



AMP Example: Pool Weight

- Lüling, Monien 1992
- Dynamic combined load balancing
- Uses the notion of pool weight (or quality):
 - Number of active nodes
 - In general, if Q_1, \dots, Q_n are the nodes stored in a pool, the weight of that pool is given by:
$$\sum_{i=1}^n (Z^u - Z^l(Q_i))^p$$
 - If $p = 0$, this measure corresponds to the number of active nodes
 - $p = 2$ was used in the experiments: interesting if the lower bounds are distributed in large intervals



AMP Example: Search Strategies

- Clausen, Perregaard 1999
- Compares four search strategies in sequential and parallel
 - Lazy versus eager
 - Best-First versus Depth-First
- Tests on bounding procedures where preprocessing (variable fixing) is performed at each node
- Depth-First outperforms Best-First in parallel and sequential: effect of preprocessing!
- Lazy tends to be better



Lessons from Past Experience

- Synchronization appears unnecessary
- ASP algorithms are appropriate:
 - For problems with time-consuming bounding operations
 - For parallel systems with few processors (scalability issue)
- Use concurrent data structures in ASP algorithms
- Dynamic load balancing: a must in AMP algorithms
 - Combined strategies appear most promising
 - Use the weight of the node pools
- Mixed organization (global pool + local pools): an interesting alternative to collegial organization

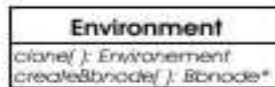
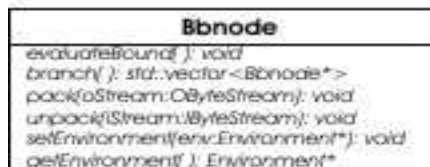


Software Tools

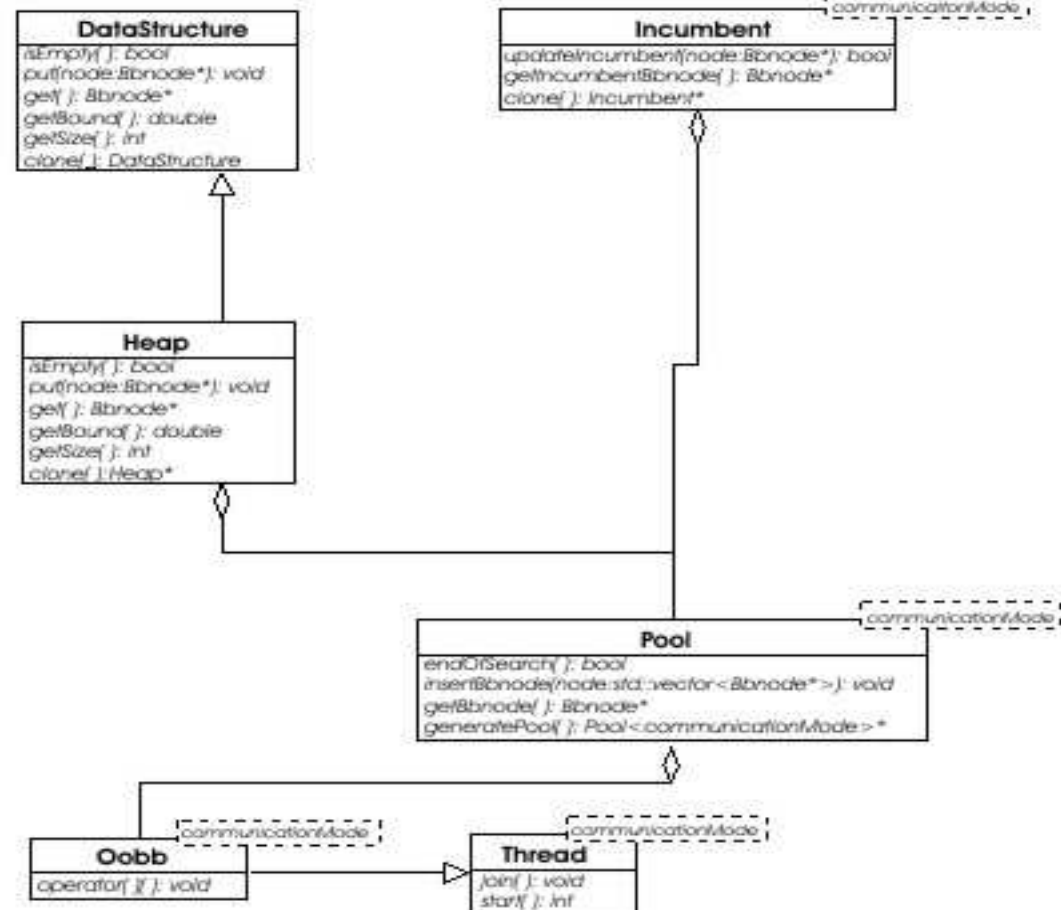
- Bob++ (Roucairol, Le Cun et al., U. Versailles)
 - Shared-memory environments
 - Integration of other search algorithms (A*, DP,...)
- PICO (Eckstein et al.)
 - Message-passing environments
 - C++ with MPI and homemade threads
- COIN/BCP, ALPS, BiCePS (Ralphs, Ladanyi, Saltzman)
 - Message-passing and shared-memory environments
 - C++ with PVM, MPI, OpenMP
 - ALPS: Parallel Search; BiCePS: constraint/variable generation
- OOB (Crainic, Frangioni, Gendron, Guertin)
 - Message-passing and shared-memory environments
 - C++ with MPI and PosixThreads

OBB: Class Organisation

User Interface



Parallel Interface





OOBB: Example of *main*

- Declare *communicationMode*

```
//#define communicator Sequential  
#define communicator ComThread  
//#define communicator ComMpi
```

- Create the first node

```
BBN_TSP *node = new BBN_TSP(tsp_dat_file);
```

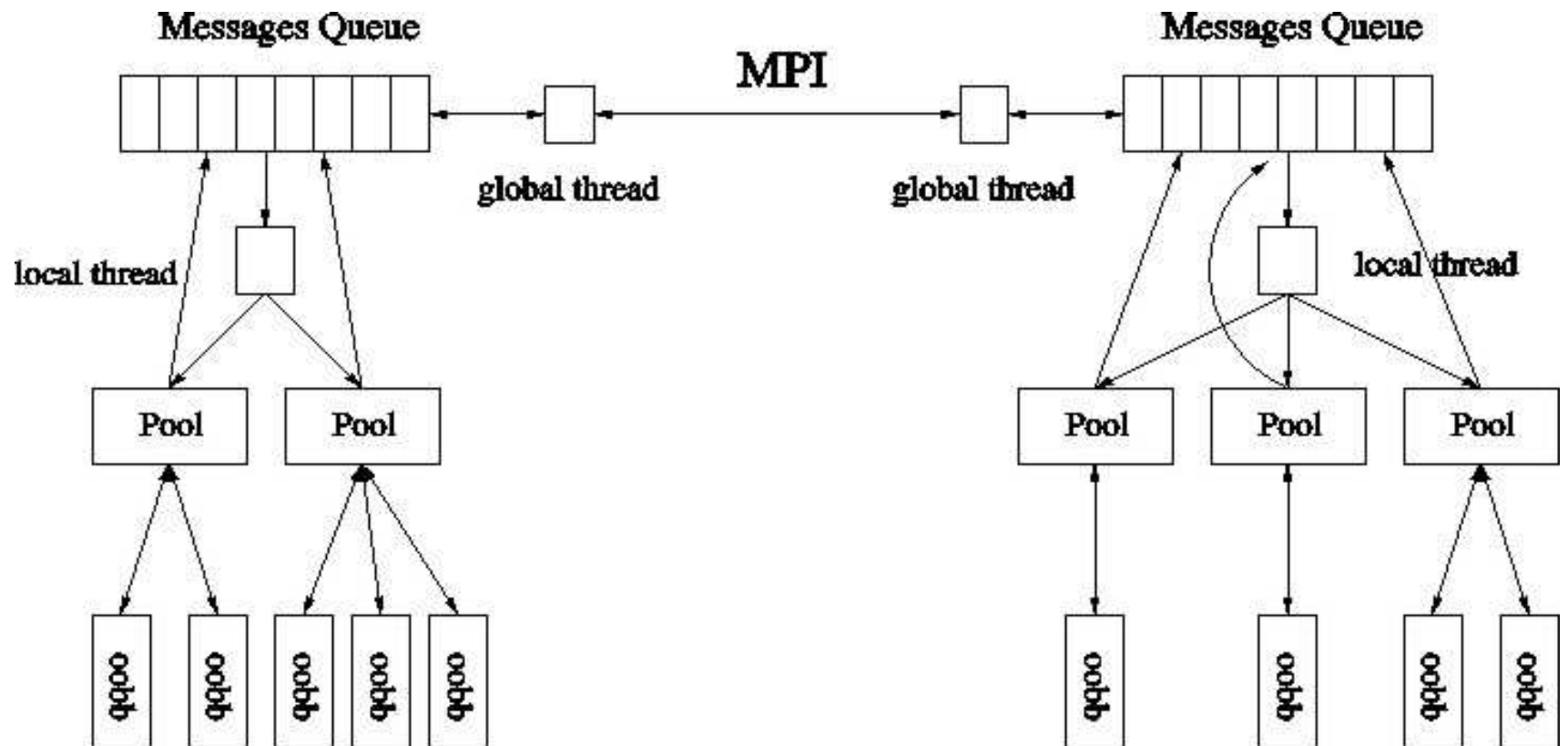
- Create other objects

```
Incumbent<communicator> *incumbent = new Incumbent<communicator>(node, environmentTsp);  
Pool<communicator> *pool = new Pool<communicator>(node, incumbent, parameter, environmentTsp);  
Oobb<communicator> *oobb = new Oobb<communicator>(parameter);
```

- Run B&B

```
solve(pool, oobb, environmentTsp);
```

OBB: Communications





Research Issues

- Combine the three types of parallelism
- Initialization strategies
- Global Dynamic Information:
 - Best upper bound: a special case
 - Pools of constraints/variables in B&C + B&P
- Adapt the general tools to GRID Computing
 - Configuration: CPUs are added at run-time
 - Fault tolerance