Parallel Computing in Combinatorial Optimization

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

- Objective: provide an overview of the current research on the design of parallel algorithms and software tools for solving combinatorial optimization problems
- Methods: tabu search, simulated annealing, variable neighborhood search, genetic algorithms, ant systems, branch-and-bound, constraint programming
- Various parallel and distributed environments
- Applications: TSP, VRP, QAP, network design
Calendar

- Two lessons/week for three weeks
- Lesson 1: overview
- Lesson 2: parallel metaheuristics
- Lesson 3: parallel branch-and-bound
- Lesson 4: parallel constraint programming + hybrid
- Lesson 5: applications
- Lesson 6: presentation of papers by the students
Presentation of a Paper

- At the end of the first week, all papers will be listed on the course Web site.
- At the beginning of Lesson 3, each student will communicate his **three preferred papers** to the Professor.
- At the beginning of Lesson 4, each student will be assigned a paper to present.
- The presentation will take place during Lesson 6.
Short Paper

- Write a short paper (about 5 pages) to be submitted to the Professor no later than August 1
- The subject could be:
  - A short survey on a specific aspect studied during the course, or...
  - A proposal about a potential research project
- The subject should be communicated to the Professor at the beginning of Lesson 5 for subsequent approval
Characteristics of Parallel Systems

- Control
  - Flynn’s classification: SIMD and MIMD systems

- Synchronization
  - Synchronous versus asynchronous systems

- Communication
  - Shared-memory versus message-passing systems

- Grain
  - Fine-grained versus coarse-grained systems

- Number of processors
  - Small-scale versus large-scale (massively parallel) systems
Design of Parallel Systems: Tendencies

- Loose interconnection, no more rigid topology
- Networks of workstations/PC
- *Grid computing*: large-scale distributed environments
- Most common programming environments:
  - Shared-memory: PosixThreads, OpenMP
  - Message-passing: PVM, MPI
- Virtual programming environments: better portability
Parallel Performance Measures

- Elapsed time with $p$ processors: $T(p)$
- Speedup: $S(p) = T(1)/T(p)$
  - Ideally, $T(1)$ is not the time taken by the parallel algorithm on 1 processor, but the time required by the “best” sequential algorithm
  - Normally, $1 \leq S(p) \leq p$
- Efficiency: $E(p) = S(p)/p$
- Overhead: $O(p) = T(p)(1-E(p)) = T(p)-(T(1)/p)$
- Scalability: how should the size of the problem vary in order to maintain constant efficiency irrespective of the number of processors used?
Combinatorial Optimization (CO)

- \( \min \{ f(x) \mid x \in S \} \)
  - \( f \): objective function
  - \( f(x) \): value of solution \( x \)
  - \( S \): set of feasible solutions
  - In CO, \( S \) is a finite, but very large-scale, set

- Some well-known examples:
  - *Linear programming*: \( S \) is the set of extreme points of the polyhedron defined by the linear program
  - *Minimum spanning tree*: belongs to \( \mathbf{P} \)
  - *Traveling salesman problem*: \( \mathbf{NP} \)-hard
For \textbf{NP}-hard CO problems, the only exact algorithms we know are based on enumeration.

The worst-case complexity of these algorithms is either

- Pseudo-polynomial (for instance, the classical dynamic programming algorithm for the 0-1 knapsack problem)
- Exponential (branch-and-bound algorithms for the same problem)

Alternative: heuristic algorithms
Heuristic Algorithms

- **Constructive heuristic**: builds a feasible solution from scratch (for example, a greedy-like algorithm)
- **Improvement heuristic**: starting from a feasible solution, the heuristic attempts to improve its value
  - Often based on the notion of *neighborhood*: given a feasible solution, neighbor solutions are obtained by performing *movements* of a certain type
- **Metaheuristic**: general framework for developing heuristics for specific problems
Neighborhood-Based Metaheuristics

- **Descent**: the simplest of these so-called *local search* metaheuristics
  - Start from some initial solution (for example, obtained by a constructive heuristic)
  - If there is no solution in the neighborhood of the current solution that improves the value of the objective, STOP
  - Otherwise, move to a solution in the neighborhood that improves the value of the objective and iterate

- **Difficulties with this metaheuristic**:  
  - Dependent on the initial solution  
  - Trapped into local optimal solution
Tabu Search

- Metaheuristic proposed by Fred Glover (1986) to avoid the pitfalls of the descent method
- Idea: perform a movement even if it does not improve the value of the objective
- More precisely, pick the best solution in the neighborhood even if it is not improving
- But in this case, we need to avoid going back to the previously visited solutions
- Idea: keep in a short-term memory, called tabu list, the reverse movements that led to the last visited solutions
Other Ingredients in Tabu Search

- Forbid these tabu movements unless one of them yields a better solution than the currently best one (*aspiration* criterion)
- In addition to the short-term tabu list, one should also use long-term memories that retain global information about the whole search process to
  - Perform some form of *intensification*: explore more intensively areas close to the « best » solutions
  - Perform some form of *diversification*: explore regions of the solution space which have not been visited so far
Sources of Parallelism in Tabu Search

- Obvious: neighborhood evaluation
- Also obvious: perform several tabu searches *independently* in parallel, starting with different initial solutions
- Less obvious: perform several tabu searches in parallel, that *cooperate* with each other
  - Communicate information between these tabu searches to perform a more efficient global search
  - Objective: when increasing the number of processors, obtain better solutions with the same amount of work!
Branch-and-Bound (B&B)

- Exact algorithm based on divide-and-conquer
- **Branching**: If the problem cannot be solved directly, partition the set of solutions, $S$, into $q$ subsets, $S_1, S_2, ..., S_q$
- Try to solve each subproblem: $\min \{ f(x) \mid x \in S_j \}$
- If it cannot be done directly, partition each subset $S_i$ in a similar way
- When all subproblems are solved, the minimum value corresponds to optimal value of the original problem
This divide-and-conquer process can be represented by the construction of a tree:
- The root corresponds to the original problem.
- Each node corresponds to a subproblem.
- Each leaf corresponds to a solved subproblem.
- Each node has one parent and several children (unless it is a leaf).

To speedup this enumeration process and prune (or fathom) large parts of the tree, we compute lower and upper bounds on the optimal value of each subproblem.
Bound Computations in B&B

- $Z^l(T)$: lower bound on the optimal value of subproblem $T$
- $Z^u(T)$: upper bound on the optimal value of subproblem $T$
- $Z^l(T) \geq Z^l(P(T))$ where $P(T)$ is the parent of $T$
- $Z^u(T)$ is the objective value of a feasible solution for subproblem $T$, hence a feasible solution for the original problem
- $Z^u(S)$: lowest upper bound found so far
- **Pruning by bound:** if $Z^l(T) \geq Z^u(S)$, $T$ is pruned
Prototype B&B Algorithm

1. $L = \{S\}$ (L is the list of *active nodes*); $Z^u(S) = +\infty$
2. If $L$ is empty, STOP; otherwise pick a node $T$ in $L$ and remove it from $L$ (*selection*)
3. Compute lower and upper bounds: $Z^l(T)$ and $Z^u(T)$
4. If $Z^u(S) > Z^u(T)$, $Z^u(S) = Z^u(T)$ and keep in memory the corresponding best feasible solution
5. If $Z^l(T) \geq Z^u(S)$, go to 2
6. Perform branching on $T$, add the new nodes to $L$ and go to 2
Sources of Parallelism in B&B

- Obvious, but not so easy to realize: perform tree search in parallel
- Problem: search anomalies
- Crucial: how to initialize the parallel exploration?
- Other sources of parallelism:
  - Bounding and branching operations: problem-dependent
  - Speculative parallelism: perform several tree searches