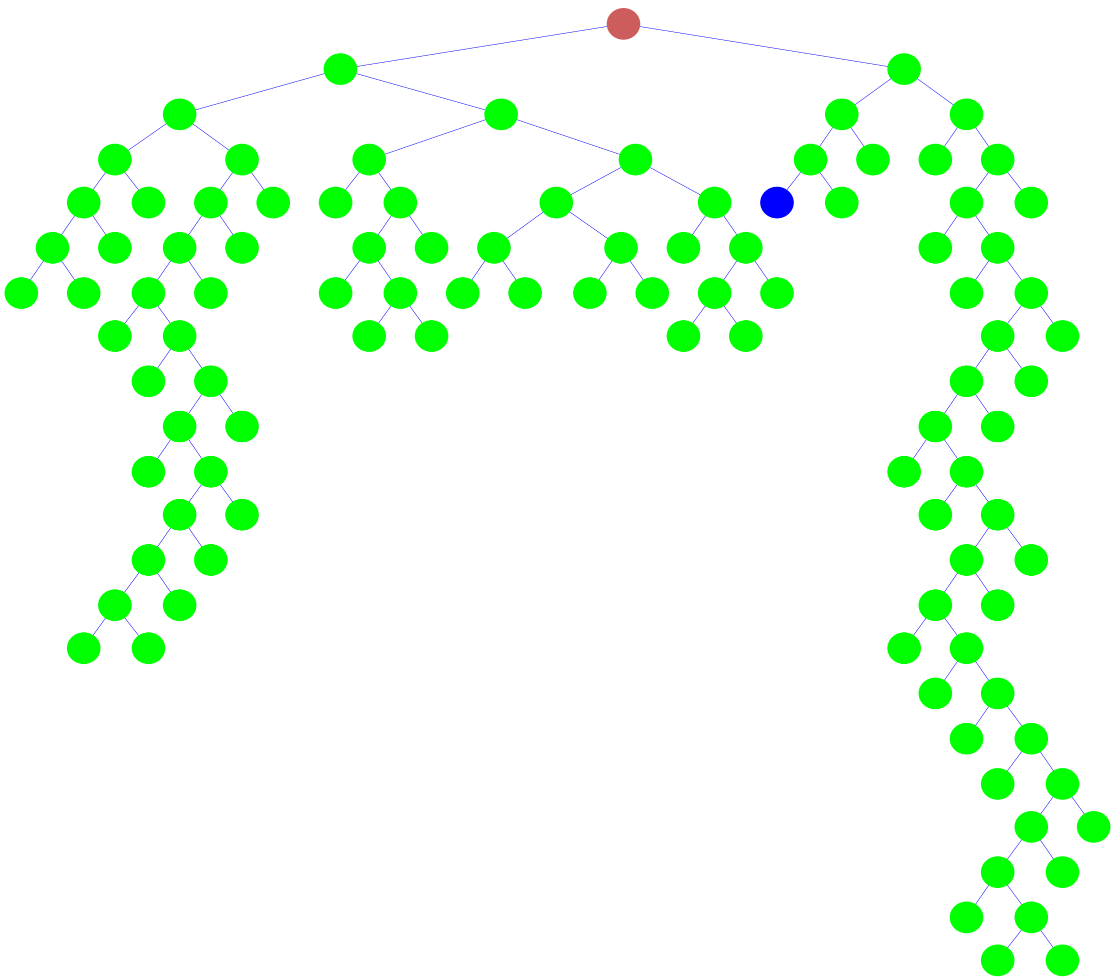
Parallel Branch and Cut for Dummies



Ted Ralphs
Department of Industrial and
Manufacturing Systems Engineering
Lehigh University
Bethlehem, PA

Outline of Talk

- Introduction to Parallel Systems
- Introduction to Tree Search
- Parallelizing Branch and Bound
- Parallelizing Branch and Cut
- The SYMPHONY Library
- Computational Results
- Conclusions

Parallel Systems

- Parallel System: Parallel algorithm + parallel architecture [Kumar and Gupta '94].
- Scalability: How well a parallel system takes advantage of increased computing resources.
- Fixed problem size: Efficiency decreases with more processors (Amdahl's Law) [Amdahl '67].
- Fixed number of processors: Efficiency increases with problem size.
- Isoefficiency: The rate problem size must be increased to maintain a fixed efficiency [Kumar and Rao '87].
- Terms

Sequential runtime	T_s
Sequential fraction	s
Parallel runtime	T_p
Parallel overhead	$T_o = NT_p - T_s$
Speedup	$S = T_s/T_p$
Efficiency	$E = S/N$

Tree Search

- Types of Tree Search Problems
  - Simple search
  - Complete search
  - Optimal search (DOPs)

- Application Areas
  - Discrete Optimization
  - Artificial Intelligence
  - Game Playing
  - Theorem Proving
  - Expert Systems

- Elements of Search Algorithms
  - Node splitting method (branching)
  - Search order
  - Pruning rules

Scalability Factors for Tree Search

- General algorithmic factors
  - Serial fraction
  - Degree of concurrency
  - Parallel overhead
  - Unnecessary/duplicate work
  - Bottlenecks

- Factors specific to search algorithms
  - The ratio of U_{calc} to U_{comm}
  - Overall size of the search tree
  - Number of branches/children per node
  - Time to "ramp up"
  - Work Distribution Scheme
    – Single work pool
    – Multiple work pools

Parallel Branch and Bound

Finding a feasible solution quickly is important.

- Diving strategies
  - Lower memory and communication requirements.
  - Less node set-up time.
  - Find feasible solutions quickly.
  - Wasted computation

- Best-first strategies
  - High memory and communication requirements
  - Additional node set-up time
  - Inhibit finding feasible solutions
  - Minimize the size of the tree

- Hybrid strategies
  - Controlled Diving (Cost vs. Fractional)
  - Heuristics (initial and primal)

Parallel Branch, Cut, and Price

Global pools are an important additional factor.

- Node pools
  - Single pool
    - easy maintenance, load balancing, and termination detection
    - more accurate "global picture"
    - can become a bottleneck
  - Multiple pools
    - increase communication requirements
    - difficult to load balance
    - reduce "saturation"

- Cut and Column Pools
  - Must be scanned linearly
  - Single pool
    - slow to scan when large
    - (possibly) better global information
  - Multiple pools
    - smaller and faster to scan
    - contain more localized information
    - more overhead

Performance Measures for Parallel Search

- Measures of redundant/unnecessary work
  - number of nodes in the search tree
  - average time to process a node
- Measures of idle time
  - time spent waiting for work
  - time spent waiting for cuts/columns from pool

SYMPHONY

SYMPHONY is a generic framework for implementing parallel branch, cut, and price.

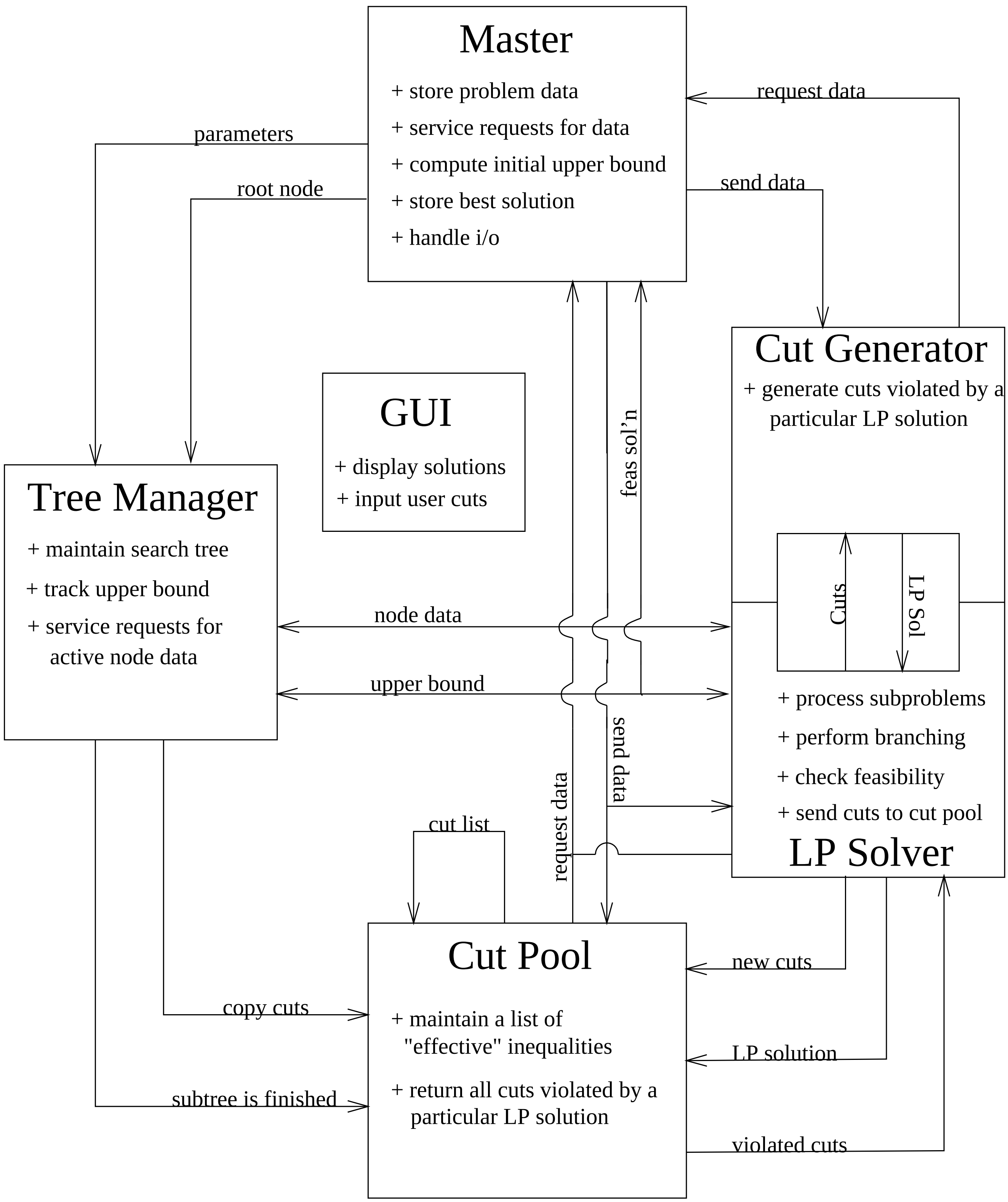
- It was designed specifically to run in parallel.
  - Can be built for serial, distributed, or shared-memory environments.
  - No knowledge of parallelism needed.
  - Runs on multiple platforms—even mixed.

- User supplies:
  - separation subroutines,
  - the initial LP relaxation,
  - feasibility checker, and
  - other optional subroutines.

- SYMPHONY takes care of all other aspects of algorithm execution, including communication.

- Available at `www.BranchAndCut.org`.

The Processes of Parallel Branch and Cut



Branching

- Branching decisions are critical, especially near the top of the tree.
- We want to minimize "ramp up" time while not increasing overall execution time – difficult!!
- Can branch on cuts or variables.
  - Any number of children is allowed.
  - Branch on several left hand values for a constraint.
- Strong branching is indispensable.
  - Select several branching candidates (can be cuts, variables, or both).
  - "Presolve" each candidate.
  - Choose the "best" for branching.
  - More candidates are evaluated near the top of the tree.

Pool Management

- Node Storage
  - Candidate nodes stored in a single pool.
  - Try to minimize memory use while not increasing node communication time.
  - The complete state of the tree is stored.
  - Node descriptions are stored explicitly or w.r.t. parent.
- Cut Storage
  - Cuts are stored in a single pool or multiple pools in a compact, efficient form.
  - The cut pools are allocated dynamically and serve assigned subtrees.
  - Pools are purged when they become too large.
- Search Management
  - The search algorithm is a best-first search with controlled diving based on either cost or degree of fractionality.
  - Best-first allows selection of "best" node.
  - Diving avoids node set-up costs and allows feasible solutions to be found quickly.

Summary Results: Vehicle Routing Problem

- Tests were performed on a network of 3 workstations powered by 4 DEC Alpha processors each. The problems are easy- to medium-difficulty problems from VRPLIB and other sources.
- Idle time is the total time spent waiting for work.
- Cut generation was not parallelized and no initial upper bound was provided for these runs.
- Software was SYMPHONY 2.7b and CPLEX 6.5.

	Number of LP processes			
	1	2	4	8
# of nodes	6593	6691	6504	6423
WC time	2493	1281	666	404
WC/node	0.38	0.38	0.41	0.50
Idle time	12.68	66.97	208.34	497.45
Idle/node	0.00	0.01	0.03	0.08

Summary Results: Traveling Salesman Problem

- Tests were performed on a network of 3 workstations powered by 4 DEC Alpha processors each. The problems are easy- to medium-difficulty problems from TSPLIB.
- Idle time is the total time spent waiting for work.
- Cut generation was not parallelized and no initial upper bound was provided for these runs.
- Software was SYMPHONY 2.7b and CPLEX 6.5.

	Number of LP processes			
	1	2	4	8
# of nodes	3405	3041	2917	2901
WC time	18119	10057	7445	5199
WC/node	5.32	6.60	10.21	14.34
Idle time	24.34	935.42	3527.61	10767.89
Idle/node	0.01	0.31	1.21	3.71

Another Experiment

Comparison of full diving to controlled diving.

	Number of LP processes		
	1	2	4
nodes	5197	4224	4091
WC	23695	12558	6912
WC/node	4.56	5.95	6.76
nodes	3405	3041	2917
WC	18119	10057	7445
WC/node	5.32	6.60	10.21

Further Experiments

- Diving strategy

	VRP		TSP	
	Nodes	Time	Nodes	Time
w/ UB	3524	241	2720	6585
w/o UB	6504	666	2917	7444

- Parallel Overhead

	Nodes	Time	Time/Node
Parallel	7014	24558	3.50
Sequential	7014	25446	3.63

- Shared vs. Distributed Memory

	Nodes	Time	Time/Node
Shared	8896	2975	1.34
Distributed	9701	3392	1.40

Conclusions and Future Work

- Architecture does not play a very big role.
- Ramp-up time is not a problem.
- Unnecessary/duplicate work is not an issue.
- The cut pools are not significant bottlenecks.
- The single biggest factor in loss of speedup is the bottleneck created by the tree manager.
- Multi-pool approaches are well-studied, but tough to implement (see [Eckstein '94], [Gendron and Crainic '94]).
- A single tree manager with several slave node pools might be a good compromise (ala [Eckstein '94]).
- A more decentralized approach may also be undertaken.