A Library Hierarchy for Implementing Scalable Parallel Search Algorithms

Laszlo Ladanyi
IBM T.J. Watson Research Center

Ted Ralphs
Lehigh University

Matt Saltzman
Clemson University

Broad Goals of Project

- Develop follow-on to COIN/BCP with increased functionality.
- Should be as generic as possible.
  - Support for implementing general tree search algorithms.
  - Support for any bounding scheme.
  - No assumptions on problem/algorithm type.
- Should be scalable and as general as possible.
- Should support data-intensive algorithms (BCP).
- Should support advanced methods not available in commercial codes.

Outline of Talk

- Multi-layered, modular C++ class library
- Improved scalability
- Data handling / object representation
- Enhancements (multi-phase method, decomposition, fault tolerance)

Layered design

Modular library design with minimal assumptions in each layer.

ALPS Abstract Library for Parallel Search

- manages the search tree.
- prioritizes based on quality.

BiCePS Branch, Constrain, and Price Software

- manages the data.
- adds notion of primal and dual objects.
  - dual objects are functions of primal objects.
  - objects have bounds, a value, and a price.
  - objective function is a dual object without bounds.

BLIS BiCePS Linear Integer Solver

- assumes linear constraints and a linear objective.
- utilizes LP relaxations.

ALPS: Abstract Library for Parallel Search

Properties of a search tree node:

- status: `candidate`, `processed`, `branched`, `fathomed`.
- quality: a numerical priority (below threshold $\Rightarrow$ fathomed).

Operations on the search tree nodes:

- create children (branch).
- remove a node (recursively: remove a subtree).

ALPS: Abstract notions.

Procedural abstractions:

- process
  - status `candidate` $\rightarrow$ processed/fathomed.
- branch
  - status `processed` $\rightarrow$ `branched`.
  - create children (`candidate` nodes) and add to queue.

Data management abstraction:

- differencing scheme: node description can be stored with respect to parent.
- explicit description can be extracted, relative description can be created.

BiCePS: Branch, Constrain, and Price Software

Adds the notion of objects in terms of Lagrangean duality:

- primal objects or variables have associated
  - value: must lie between the primal object's bounds.
  - reduced cost: the partial derivative of the objective function.
- dual objects or constraints are functions of the primal objects and have associated
  - value: the Lagrange multipliers.
  - slack: the distance from the bounds of the evaluated function of the dual object.
- objective: a function of the primal objects (a dual object without bounds, value, and slack).

Primal and dual solutions: the objects with their associated values.

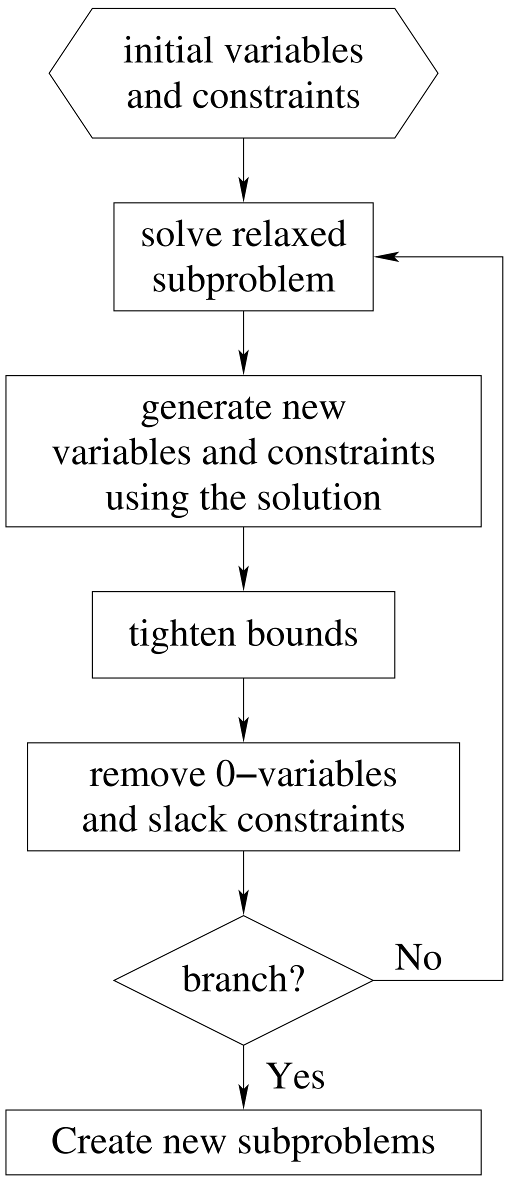
BiCePS: Processing a subproblem

A subproblem is a set of objects with an objective.

Processing a subproblem

- solve a relaxation.
- generate new objects.
- tighten bounds.
- remove objects with value 0.

If all else fails or when desired, branch.



BiCePS: Branching

Traditional branching: Choose $\hat{x}_j$ fractional.

Children: $x_j \leq \lfloor \hat{x}_j \rfloor$ and $x_j \geq \lceil \hat{x}_j \rceil$.

General branching:

- Add new objects.
- Change object bounds.
- Children must cover the feasible region.

Example: y_i binary variable and $y_i = 0 \Rightarrow a^T x \leq \beta$.

Children: $y_i = 1$ and $\{y_i = 0$ and $a^T x \leq \beta\}$.

(this avoids using the big M method)

BLIS: BiCePS Linear Integer Solver

A concretization of BiCePS specifying the bounding relaxation to be used.

Defines:

- the representation of the objects: columns/rows.
- the relaxation to be used: LP relaxation.
- the objective function: linear.

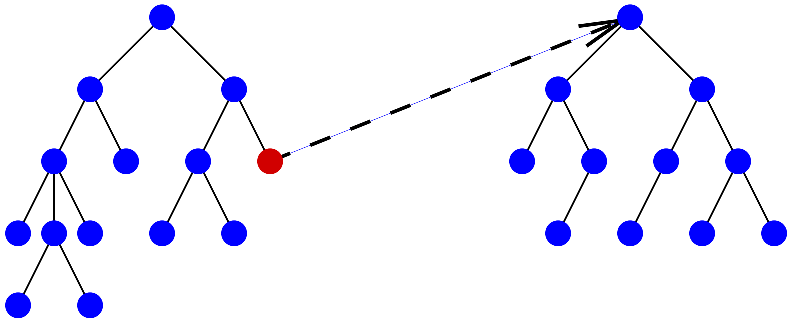
Leaves the notion of the relaxation solver abstract by using Open Solve Interface.

Scalability Issues

- Grain size
- Decentralization
- Synchronous vs. asynchronous messaging
- Ramp-up/ramp-down time

Scalability: Increased granularity

Work unit is a subtree.



Advantages:

- less communication.
- more compact storage via differencing.

Disadvantage:

- load balancing is more difficult.

Scalability: Master - Hubs - Workers Paradigm

Master

- has global information (node quality and distribution).
- balances load between hubs.
- balances quantity *and* quality.

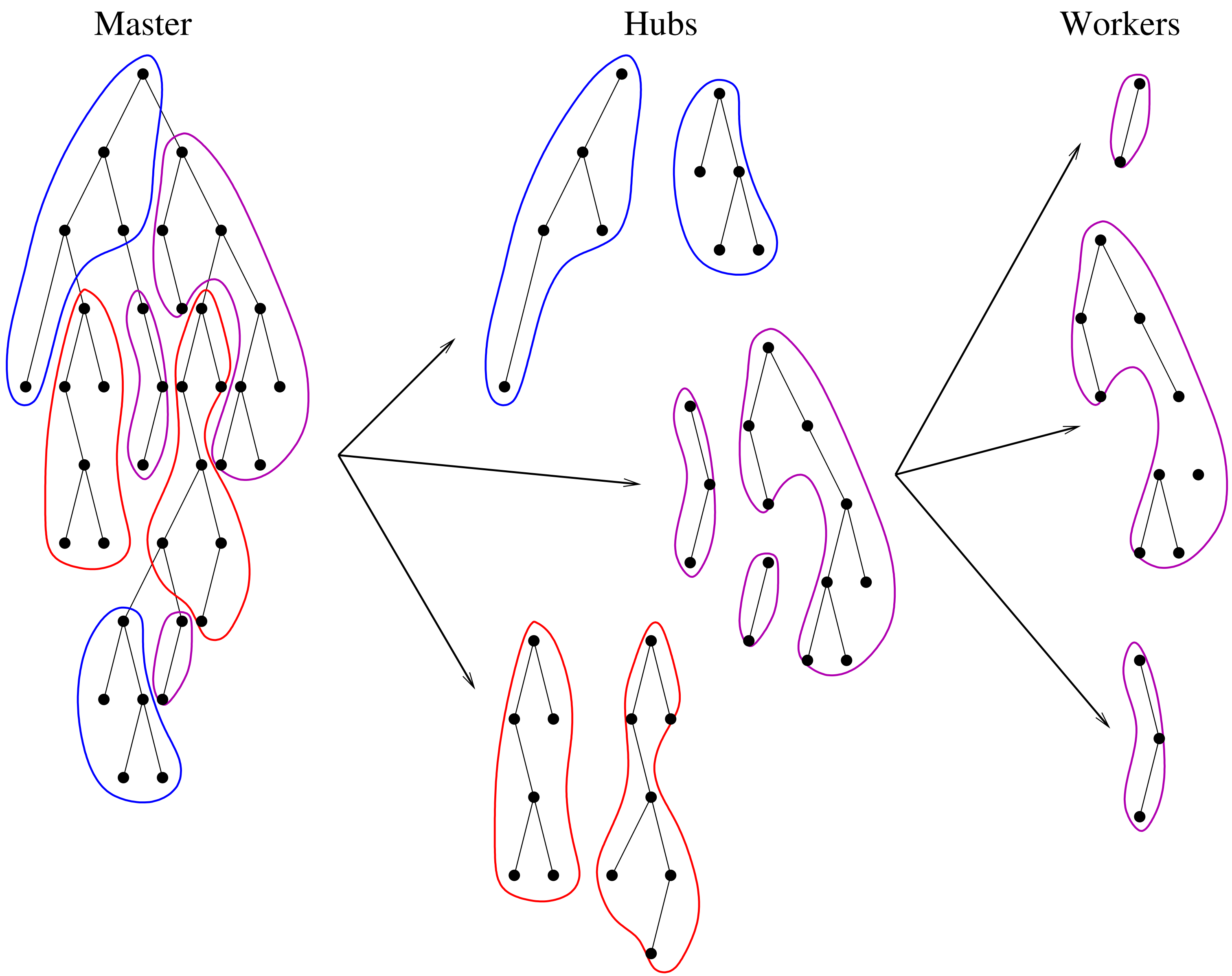
Hubs

- manage collections of subtrees (may not have full descriptions)
- balances load between workers

Workers

- processes one subtree.
- hub can interrupt.
- sends branch and quality information to hub.

Scalability: Master - Hubs - Workers Paradigm



Scalability: Asynchronous messaging

Possible communication bottlenecks:

- Too many messages.
  - avoided by the increased task granularity.
  - master-hub-worker paradigm also contributes.
- Too much synchronization (handshaking)
  - almost no handshaking.
  - must take place when a worker finishes exploring a subtree.

Scalability: Ramp-up/Ramp-down

- Ramp-up time: Time until all processors have useful work to do.
- Ramp-down time: Time during which there is not enough work for all processors.
- Controlling Ramp-up/ramp-down
  - Use different branching rules.
  - Hub instructs workers when to change rules.

Data Handling Issues

- Focused on data-intensive applications.
- Need to deal with huge numbers of objects.
- Need compact storage.
- Need to avoid duplication (generation and storage).

Data Handling: Object Representation

Each object has three representations:

- the user's representation.
  - information to generate the realization.
  - core, indexed, or algorithmic.
- the realization in the solver.
  - (projected) matrix row
  - (projected) matrix column
- the encoded representation.
  - for identification and transfer between processors.

Data Handling: Encoding / Decoding

Encodable objects: Any object that is sent between processes.

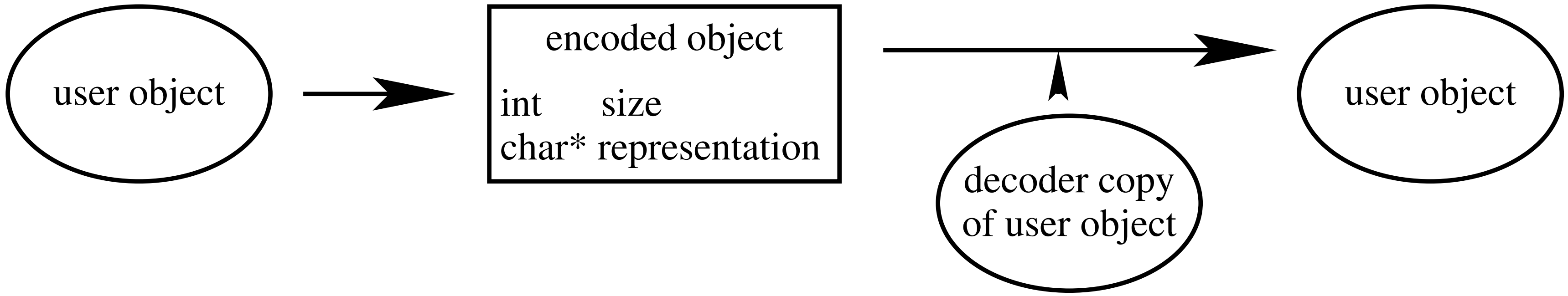
Question: How to encode/decode objects the framework does not know about?

Encoding: Easy $\Rightarrow$ use virtual methods.

Decoding: Catch-22

- can't invoke constructor for unknown type.
- can't invoke decode method without an object.

Solution: "Register" encodable objects.



Data Handling: Tracking Lists of Objects

Goal is memory conservation: no unnecessary object storage.

Implementation:

1. Object arrives in encoded form with hash value.
2. Object is looked up in hash map.
3. If it does not exist, then it is inserted.
4. A pointer to the unique copy in the hash map is added to the list.

Primary uses:

- storing variables and constraints—object can be active in multiple search nodes, but only one copy will be stored.
- object pools.

Data Handling: Object Pools

- Share objects across nodes in the tree.
- Object pools allow generated objects to be shared.
  - Cut pool is one example.
- Quality measures (slack or reduced cost, tree level, touches) ensure only the most effective objects are utilized.
- Object encoding is used to ensure that objects are stored only once.

Enhancements

- Domain decomposition
- Multi-phase methods
- Fault tolerance
- On-the-fly reconfiguration

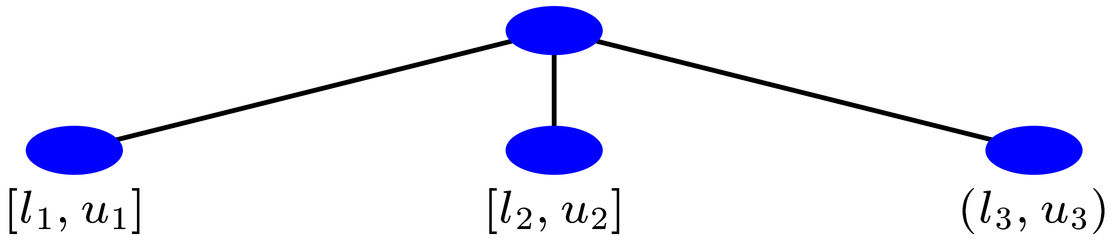
Domain Decomposition

Useful when primal and dual objects can be partitioned such that

- primal objects of a group do not interact with dual objects of other groups.
- the objective object is f-separable in terms of the primal variables in the groups. f can be additive (traditional MILP, stochastic programming), multiplicative (MIQP), etc.

After decomposing solve the children recursively, propagating intermediate results (thus creating new upper bound or fathoming the whole subtree).

$$[l, u] = [f(l_1, l_2, l_3), f(u_1, u_2, u_3)]$$



Multi-phase Methods

- Solve the problem on a subset of the variables, resulting in
  - a good upper bound,
  - a collection of good cuts,
  - an explored search tree.
- Price the remaining variables and propagate survivors down the tree, repeating pricing periodically.
- Unfathomed leaves are entered into the candidate list for the next phase.
- Difficulty: reproducing the search tree.

Fault Tolerance

Recovering from process failure (hardware or software)

- Worker $\Rightarrow$ easy.
  - only the processed subtree is lost.
  - the hub can reassign the work.
- Object pool $\Rightarrow$ easy.
  - has no effect on correctness.
  - can be restarted.
- Hub $\Rightarrow$ hard.
  - all managed subtrees are lost.
  - other hubs must discard subtrees whose parents were on the dead hub.
  - workers must be reassigned to another hub.
- Master
  - can attempt to restart from saved data.

On-the-fly Reconfiguration

- On the fly reconfiguration (planned “fault”) — restricted use of processors.
- Messages from external source instructing master to
  - offload work from a process and kill it—work is redistributed.
  - to start new processes and redistribute work.

Development Roadmap

- Finish laying out class structure (almost done).
- Write worker implementation—stand-alone sequential code.
- Interface with CGL.
- One hub $\Rightarrow$ multiple workers.
- Master $\Rightarrow$ multiple hubs $\Rightarrow$ multiple workers.
- Object pools.
- Domain decomposition.
- Multi-phase.