Solving Convex MINLPs with MINOTAUR: Presolving, Cuts and More

Ashutosh Mahajan

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay

June 04, 2014
About this talk

The Team
About this talk

The Agenda

1. Describe the Minotaur framework
2. Glimpse of what is under the hood
3. Two main themes:
   - Exploit specific problem structures
   - Customize specific components of Branch-and-Cut
4. Illustrate how you can extend or customize it
What is Minotaur?

**Mixed**

**Integer**

**Nonlinear**

**Optimization**

**Toolkit:**

**Algorithms,**

**Underestimators,**

**Relaxations.**

http://wiki.mcs.anl.gov/minotaur

- Completely open-source: BSD License
- Source code, libraries, binaries available
- Well tested on Linux, Mac
- Documentation on the wiki, examples in the source
Three ways to use Minotaur

4 Use the binaries (through AMPL or Pyomo)
   - NLP based branch-and-bound
   - LP-NLP based branch-and-bound
   - QP-diving
   - Solvers for nonconvex problems under development
   - Available as minotaur-xxx-bin-yyy.tar.gz
Three ways to use Minotaur

1. Use the binaries (through AMPL or Pyomo)
   - NLP based branch-and-bound
   - LP-NLP based branch-and-bound
   - QP-diving
   - Solvers for nonconvex problems under development
   - Available as minotaur-xxx-bin-yyy.tar.gz

2. Use the libraries
   - Use existing interfaces, methods, engines, branchers, tree-manager, ...
   - Embed it in your application
   - Available as minotaur-xxx-dev-yyy.tar.gz
Three ways to use Minotaur

1. Use the binaries (through AMPL or Pyomo)
   - NLP based branch-and-bound
   - LP-NLP based branch-and-bound
   - QP-diving
   - Solvers for nonconvex problems under development
   - Available as minotaur-xxx-bin-yyyy.tar.gz

2. Use the libraries
   - Use existing interfaces, methods, engines, branchers, tree-manager, . . .
   - Embed it in your application
   - Available as minotaur-xxx-dev-yyyy.tar.gz

3. Build your own
   - Modify/add your own code to Minotaur
   - Available as minotaur-xxx-src.tar.gz
   - C++, modular
Inside Minotaur: Three Main Components

Core

1. Problem Description Classes
   - Function,
   - NonlinearFunction,
     LinearFunction,
   - Variable, Constraint, Objective, etc.

Engines

1. OSI-LP (coin-or.org)
   - CLP
   - CPLEX
   - GUROBI

Interfaces

1. AMPL
2. C++
Inside Minotaur: Three Main Components

Core

1. Problem Description Classes
   - Function,
   - NonlinearFunction,
   - LinearFunction,
   - Variable, Constraint, Objective, etc.

2. Branch-and-Bound Classes
   - NodeRelaxer, NodeProcessor
   - Brancher, TreeManager
   - Presolver, CutManager, etc.

Engines

1. OSI-LP (coin-or.org)
2. CLP
3. CPLEX
4. GUROBI
5. IPOPT
6. Filter-SQP

Interfaces

1. AMPL
2. C++
Inside Minotaur: Three Main Components

Core

1. Problem Description Classes
   - Function,
   - NonlinearFunction,
   - LinearFunction,
   - Variable, Constraint, Objective, etc.

2. Branch-and-Bound Classes
   - NodeRelaxer, NodeProcessor
   - Brancher, TreeManager
   - Presolver, CutManager, etc.

3. Structure Handlers
   - Linear, SOS2, CxUnivar, CxQuad,
   - Multilinear, QG, etc.
## Inside Minotaur: Three Main Components

<table>
<thead>
<tr>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.</strong> Problem Description Classes</td>
</tr>
<tr>
<td>- Function,</td>
</tr>
<tr>
<td>- NonlinearFunction,</td>
</tr>
<tr>
<td>- LinearFunction,</td>
</tr>
<tr>
<td>- Variable, Constraint, Objective, etc.</td>
</tr>
<tr>
<td><strong>2.</strong> Branch-and-Bound Classes</td>
</tr>
<tr>
<td>- NodeRelaxer, NodeProcessor</td>
</tr>
<tr>
<td>- Brancher, TreeManager</td>
</tr>
<tr>
<td>- Presolver, CutManager, etc.</td>
</tr>
<tr>
<td><strong>3.</strong> Structure Handlers</td>
</tr>
<tr>
<td>- Linear, SOS2, CxUnivar, CxQuad, Multilinear, QG, etc.</td>
</tr>
<tr>
<td><strong>4.</strong> Utility Classes</td>
</tr>
<tr>
<td>- Timer, Options, Logger, Containers, Operations, etc.</td>
</tr>
</tbody>
</table>

## Engines
- OSI-LP (coin-or.org)
- CLP
- CPLEX
- GUROBI
- BQPD
- qpOASES
- IPOPT
- Filter-SQP

## Interfaces
- AMPL
- C++
Inside Minotaur: Three Main Components

**Core**

1. **Problem Description Classes**
   - Function,
   - NonlinearFunction,
   - LinearFunction,
   - Variable, Constraint, Objective, etc.

2. **Branch-and-Bound Classes**
   - NodeRelaxer, NodeProcessor
   - Brancher, TreeManager
   - Presolver, CutManager, etc.

3. **Structure Handlers**
   - Linear, SOS2, CxUnivar, CxQuad,
   - Multilinear, QG, etc.

4. **Utility Classes**
   - Timer, Options, Logger,
   - Containers, Operations, etc.

**Engines**

1. OSI-LP (coin-or.org)
   - CLP
   - CPLEX
   - GUROBI

2. BQPD

3. qpOASES

4. IPOPT

5. Filter-SQP
Inside Minotaur: Three Main Components

Core
1. Problem Description Classes
   - Function,
   - NonlinearFunction,
   - LinearFunction,
   - Variable, Constraint, Objective, etc.
2. Branch-and-Bound Classes
   - NodeRelaxer, NodeProcessor
   - Brancher, TreeManager
   - Presolver, CutManager, etc.
3. Structure Handlers
   - Linear, SOS2, CxUnivar, CxQuad,
   - Multilinear, QG, etc.
4. Utility Classes
   - Timer, Options, Logger,
   - Containers, Operations, etc.

Engines
1. OSI-LP (coin-or.org)
   - CLP
   - CPLEX
   - GUROBI
2. BQPD
3. qpOASES
4. IPOPT
5. Filter-SQP

Interfaces
1. AMPL
2. C++
Nonlinear Functions in Minotaur

Base Class: NonlinearFunction

```cpp
virtual void computeBounds(...);
virtual Double eval(...);
virtual void evalGradient(...);
virtual void evalHessian(...);
...
```

Derived Classes:

- **AMPLNonlinearFunction**
  Queries AMPL’s ASL library for the above functions

- **CGraph**
  Nonlinear “factorable” function is stored as a computational graph
Consider a function $f : \mathbb{R}^3 \to \mathbb{R}$, 

$$f = \frac{x_2}{\sin(4x_3 + x_1)} - 3x_1$$
Consider a function \( f : \mathbb{R}^3 \rightarrow \mathbb{R}, \quad f = \frac{x_2}{\sin(4x_3 + x_1)} - 3x_1 \)

Minotaur’s computational graph of \( f \) allows us to:

- evaluate \( f \) at a given point,
- obtain bounds, under-estimators and over-estimators of \( f \),
- evaluate “exact” gradient and hessian at a given point “cheaply” (Automatic Differentiation).

It is the default class for storing nonlinear functions.

Operators Supported: \(+, -, \times, /, \log, \ln, a^x, x^a, \sin, \cos, \tan, \sinh, \sin^{-1}, \ldots\)
Easy to Construct Them

Suppose we have a function $x_1^2 + x_2^2$
Easy to Construct Them

ProblemPtr p = (ProblemPtr) new Problem();
VariablePtr x1 = p->newVariable(0, 1, Binary, "x1");
VariablePtr x2 = p->newVariable(0, 1, Binary, "x2");

CGraphPtr cg = (CGraphPtr) new CGraph();
CNode *n2 = cg->newNode(2.0);
CNode *nx1 = cg->newNode(x1);
CNode *nx2 = cg->newNode(x2);

CNode *np1 = cg->newNode(OpPow, nx1, n2);
CNode *np2 = cg->newNode(OpPow, nx2, n2);
n2 = cg->newNode(OpPlus, np1, np2);

cg->setOut(n2);
cg->finalize(); cg->write(std::cout);
152 instances from CMU-IBM website
http://egon.cheme.cmu.edu/ibm/page.htm
Exploiting Structure Through Handlers

**Where can we exploit problem structure?**
- Relaxing
- Bounding
- Checking Feasibility
- Separating
- Branching
- Presolving

**Methods in Handler Class**
- `relaxNodeFull()`
- `relaxNodeInc()`
- `presolve()`
- `presolveNode()`
- `isFeasible()`
- `separate()`
- `getBranchingCandidates()`
- `branch()`
Exploiting Structure Through Handlers

Where can we exploit problem structure?

- Relaxing
- Bounding
- Checking Feasibility
- Separating
- Branching
- Presolving

Methods in Handler Class

- relaxNodeFull()
- relaxNodeInc()
- presolve()
- presolveNode()
- isFeasible()
- separate()
- getBranchingCandidates()
- branch()

Example 1: NLP Based Branch-and-Bound for Convex MINLPs

- We need a handler for \texttt{isFeasible()}, \texttt{getBranchingCandidates()} \textbf{and} \texttt{branch()} \textbf{only}
- \texttt{IntVarHandler} does all three
A Very Simple Branch-and-Bound Solver

BranchAndBound *bab = new BranchAndBound(env, p);

v_hand = new IntVarHandler(env, p);
handlers.push_back(v_hand);

e = new FilterSQPEngine(env);
rel_br = new ReliabilityBrancher(env, handlers);
rel_br->setEngine(e);

nproc = new BndProcessor(env, e, handlers);
nproc->setBrancher(rel_br);
bab->setNodeProcessor(nproc);

nr = new NodeIncRelaxer(env, handlers);
nr->setEngine(e);
bab->setNodeRelaxer(nr);
bab->solve();
Enhancing Branch-and-Bound

Example 2: NLP Based Branch-and-Bound for Convex MINLPs with Presolving

- **IntVarHandler** for `isFeasible()`, `getBranchingCandidates()` and `branch()`
- **LinearHandler and NlPresHandler** for `presolve()`
Enhancing Branch-and-Bound

Example 2: NLP Based Branch-and-Bound for Convex MINLPs with Presolving

- **IntVarHandler** for `isFeasible()`, `getBranchingCandidates()` and `branch()`
- **LinearHandler** and **NlPresHandler** for `presolve()`

Basic functions in presolve:
- Tighten bounds on variables and constraints.
- Fix/remove variables.
- Identify and remove redundant constraints.
- Check duplicacy.

Advanced functions in presolve:
- Improve coefficients.
- Derive implications and conflicts.
- Quadratic binary to linear
Perspective Reformulation

Recall Nick Sawaya’s talk yesterday

If $g(x) \leq 0$ is a constraint in a MINLP, and the single variable $y$ forces all $x$ to zero, and $g(0) = 0$, then the constraint can be replaced by

$$yg \left( \frac{x}{y} \right) \leq 0$$
Recall Nick Sawaya’s talk yesterday

If \( g(x) \leq 0 \) is a constraint in a MINLP, and the single variable \( y \) forces all \( x \) to zero, and \( g(0) = 0 \), then the constraint can be replaced by

\[
yg \left( \frac{x}{y} \right) \leq 0
\]
Recall Nick Sawaya’s talk yesterday

If \( g(x) \leq 0 \) is a constraint in a MINLP, and the single variable \( y \) forces all \( x \) to zero, and \( g(0) = 0 \), then the constraint can be replaced by

\[
y g\left(\frac{x}{y}\right) \leq 0 \quad \rightarrow \quad (y + \epsilon) g\left(\frac{x}{y+\epsilon}\right) \leq 0
\]
Easy to Implement (≈ 70 lines)

ynode = cg->newNode(y);
anode = cg->newNode(eps);
ynode = cg->newNode(OpPlus, anode, ynode);

// visit all nodes that have variables in them
for (it = cg->vars_.begin(); it!=cg->vars_.end();
    ++it) {
    v = *it;
    if (v != z) {
        mit = cg->varNode_.find(v);
        dnode = mit->second;
        cg->varNode_.erase(mit);

        vnode = cg->newNode(v);
anode = cg->newNode(OpDiv, vnode, ynode);

        // set parents of anode.
        ...
    }
}
Example 3: LP/NLP Based Branch-and-Bound (QG)

- **IntVarHandler** for `isFeasible()`, `getBranchingCandidates()` **and** `branch()`
- **QGHandler** for `isFeasible()`, `relaxNodeInc()`, **and** `separate()`
- Two engines, LP and NLP
- We need a different node processor

```
Solve relaxation

lb ≥ ub? yes → Return
    no → Is solution feasible?
        yes → Update ub
        no → Branch

Is solution feasible? yes → Update ub
    no → Branch

lb ≥ ub? yes → Return
    no → Is solution feasible?
        yes → Update ub
        no → Find cuts

Find cuts yes → Re-solve?
    no → Branch

Re-solve? yes → Return
    no → Update ub
```
Example 4: QP-Diving

- We solve a QP at each node (good warm-starting)
- Occasionally solve NLP to find a better estimate of active constraints
- The constraints are linearizations of active constraints
- The QP objective is the gradient of the objective added to the hessian of Lagrangian

IntVarHandler for `isFeasible()`, `getBranchingCandidates()` and `branch()`

QPDPProcessor for processing the nodes (different fathoming rules)

QPDRelaxer for creating/updating QP approximations
Closing Remarks

- Minotaur is a flexible object-oriented open-source framework
- Its components can be combined to create powerful solvers …
- … at least for convex MINLPs
- We are working on some algorithms for nonconvex MINLPs also
- We would like YOU to try implement your ideas in Minotaur
- Latest ‘Nightly’ version of the source is available on the website
- An new stable version of binaries and libraries will be available soon

http://wiki.mcs.anl.gov/minotaur