Workshop on Mixed-integer nonlinear optimization and MINOTAUR: A MixedInteger Nonlinear Optimization Toolkit

Prashant Palkar, Meenarli Sharma,
and Prof. Ashutosh Mahajan

Industrial Engineering and Operations Research
Indian Institute of Technology Bombay

51^{st} Annual Convention of ORSI and International Conference
December 16-19, 2018

Outline

(I) Overview

- Mixed-integer nonlinear programs: modeling, applications and algorithms
- MINOTAUR: architecture, plugins, engines, interfaces
- MINOTAUR compilation and capabilities (demo only)

(II) Hands-on

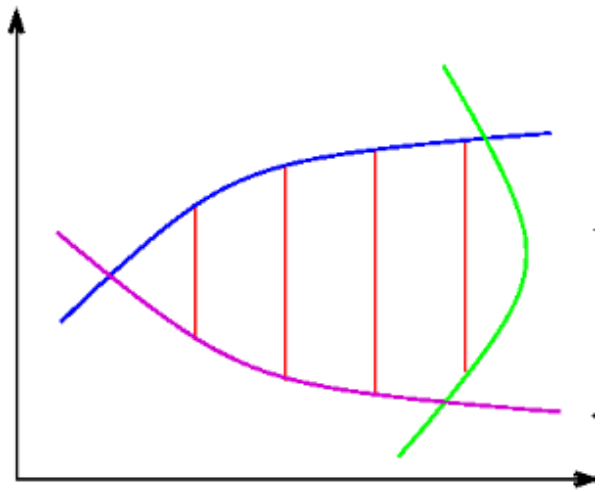
- Using different algorithms and options
- Generating problem instance: objective, variables, constraints
- Modifying existing components
- Creating a new simple brancher

Modeling mixed-integer nonlinear programs (MINLPs)

MINLPs are a general form of optimization problems. Mathematically,

$$\begin{aligned} \min_{x,y} \; & f(x,y) \\ \text{s.t.} \; & c(x,y) \leq 0, \\ & x \in \mathcal{X}, y \in \mathcal{Y} \text{ integer}, \end{aligned} \tag{P}$$

where the functions $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are typically nonlinear.



- LP, NLP, and MILP are special cases.
- If feasible region is convex on relaxing integrality, then (P) is a **convex MINLP**.

Applications and research areas

Applications

- conventional: cutting stock, portfolio optimization, facility layout, process design, unit commitment, water and gas networks etc.
- others: cybersecurity, brachytherapy, energy management, statistics, cloud, supercomputers, environment, weapons target assignment etc.

Research

- conventional: algorithms, relaxations, cuts, branchers, heuristics, presolving, structure exploitation, duality etc.
- others: representability, parallelism, overlaps with new areas: DFO, PDEs, ML, bilevel etc.

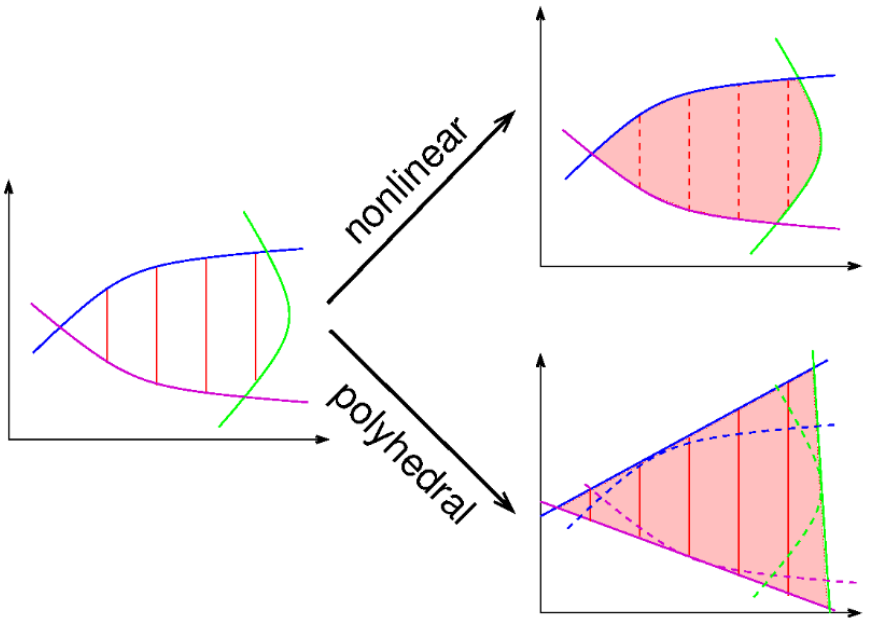
Algorithms for convex MINLPs

Basic idea

- generate lower bounds on the optimal value using *tractable* relaxations of (P)
- generate upper bounds using feasible solutions of (P)
- keep improving both until a stopping criterion is met

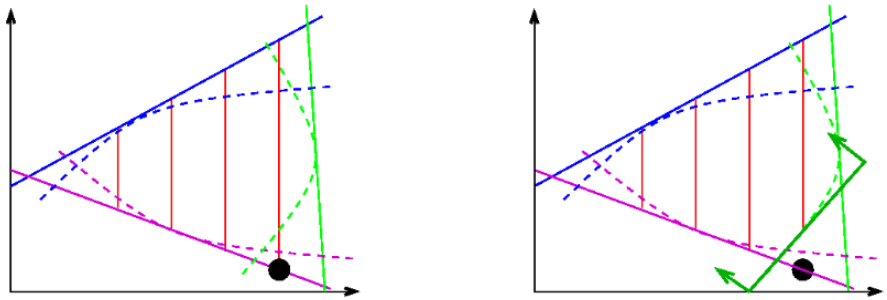
Algorithms for convex MINLPs

- Outer approximation, Nonlinear branch-and-bound, LP/NLP based branch-and-bound, Extended cutting plane etc.

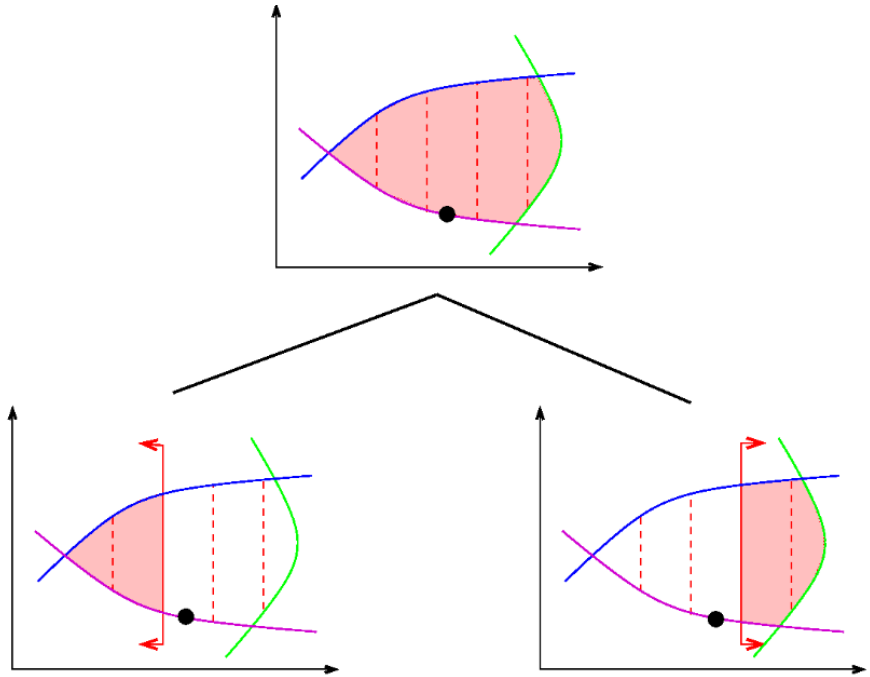


Algorithms for convex MINLPs contd.

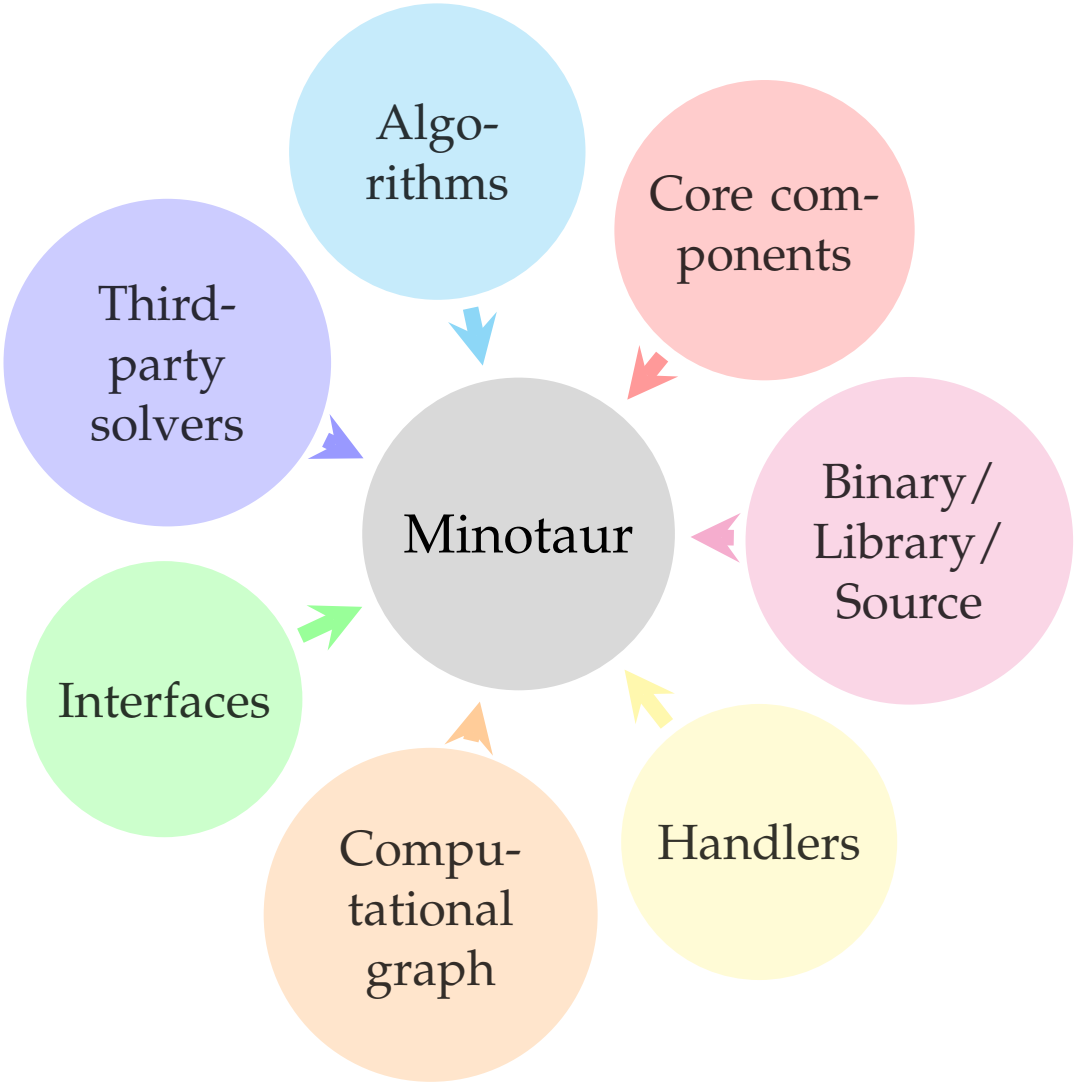
Outer approximation



Branch-and-bound



MINOTAUR (Mahajan et al, 2011)



Developed at Argonne National Laboratory, IIT Bombay and University of Wisconsin-Madison.

MINOTAUR: building blocks

Core components

- Problem Description Classes
  - Function
  - NonlinearFunction
  - LinearFunction
  - Variable, Constraint, Objective
- Branch-and-bound Classes
  - NodeRelaxer, NodeProcessor
  - Brancher, TreeManager
  - Presolver, CutManager, etc.
- Structure Handlers
  - Linear, SOS2, CxUnivar, CxQuad, Multilinear etc.
  - QG, Perspective, Separability etc.
- Utility Classes
  - Timer, Options, Logger, Containers, Operations, etc.

Engines

Linear

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

Nonlinear

- Filter-SQP
- IPOPT
- BQPD
- qpOASES

Interfaces

- AMPL
- C++

MINOTAUR compilation

On any linux system:
see complete intructions here

- GitHub page https://github.com/minotaur-solver/minotaur.git

On BITS lab machine: A precompiled version is made available.

- Open a Terminal, and type:
- `wget http://10.119.2.11/~meenarli/orsi2018/minotaurSetup.sh`
- `source minotaurSetup.sh`
- Test run: `bnb /home/student/minotaur/examples/multilinear/ex00.nl`

Hands-on

Using MINOTAUR Options

Algorithms: `bnb` and `qg`

```
bnb ~/minotaur/examples/orsi2018/instances/tls2.nl -presolve 0
```

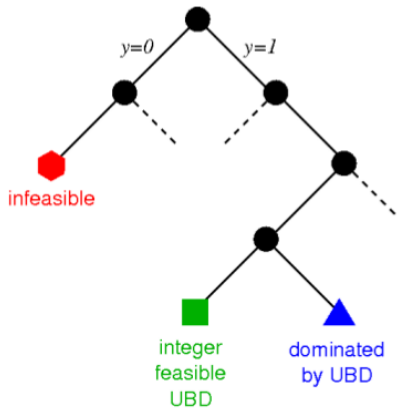
Option	Command	Value
NLP engine	`-nlp_engine`	Filter-SQP (default), IPOPT
Presolve	`-presolve`	1,0
Time limit	`-bnb_time_limit`	Time in seconds
Node limit	`-bnb_node_limit`	Any positive integer
display problem	`-display_problem`	0,1
display presolved problem	`-display_presolved_problem`	0,1
solve	`-solve`	0,1
log level	`-log_level`	0-6
brancher	`-brancher`	rel (default), lex, maxvio, etc.

Other Minotaur options here

Quick revision of NLP based branch-and-bound algorithm

Branch-and-bound

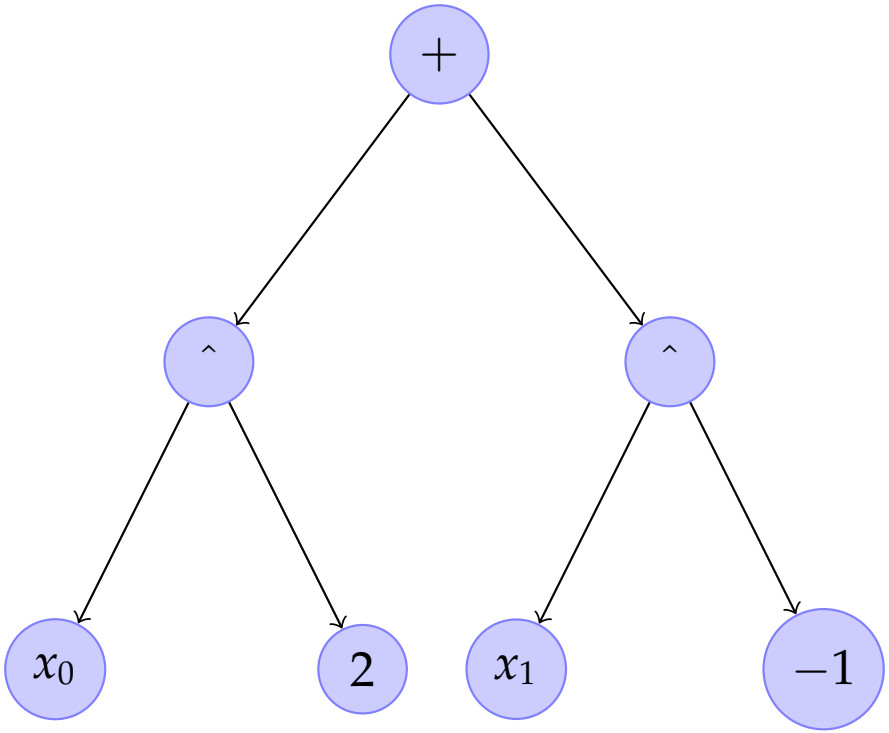
- Form a *tractable* relaxation of (P): e.g. relaxing integrality constraints gives rise to an NLP relaxation.
- If the solution of this relaxation is not integer feasible, branch on some variable in the set, $\mathcal{I}$, with fractional solution value.
- Again, *relax* the subproblems, solve them and create new subproblems by branching, if needed.
- Update the upper bound when feasible solutions are obtained and *prune inferior* subproblems.



Computational graph

Representation of nonlinear functions as a directed acyclic graph for computational purposes.

An example: $f(x) = x_0^2 + x_1^{-1}$



Problem instance generation and solving

Consider the problem

$$\begin{aligned} Min\ & x_0^{-1} + 2x_1^{-2} + 3x_2^{-1.5} + 4x_3^{-1.7} + 5x_4^{-1.2} + 6x_5^{-1.7} + 7x_6^{-1.4} + 8x_7^{-1.2} + 9x_8^{-1.5} \\ s.t.\ & x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 64 \\ & 1 \leq x_i \leq 64,\ \text{integer}\ \ \forall i = 0, \ldots, 8. \end{aligned}$$

Let us try!

- Type: `cd ~/minotaur/examples/orsi2018`
- Open the file `SimpleBnb.cpp` (**type:** `gedit SimpleBnb.cpp`)
- Go to the main function
  - function to create problem
  - solve function
- Close the file, type: `make` and to run, type: `./sbnb`
- Add a constraint: Sum of x_2 and x_6 should not be more than 2.

- Close the file, type: `make` and to run, type: `./sbnb`

More test instances on MINLPLib: A Library of Mixed-Integer and Continuous Nonlinear

Problem instance generation and solving

Consider the problem

$$\begin{aligned} Min\ & x_0^{-1} + 2x_1^{-2} + 3x_2^{-1.5} + 4x_3^{-1.7} + 5x_4^{-1.2} + 6x_5^{-1.7} + 7x_6^{-1.4} + 8x_7^{-1.2} + 9x_8^{-1.5} \\ s.t.\ & x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 64 \\ & 1 \leq x_i \leq 64,\ \text{integer}\ \ \forall i = 0, \ldots, 8. \end{aligned}$$

Let us try!

- Type: `cd ~/minotaur/examples/orsi2018`
- Open the file `SimpleBnb.cpp` (**type:** `gedit SimpleBnb.cpp`)
- Go to the main function
  - function to create problem
  - solve function
- Close the file, type: `make` and to run, type: `./sbnb`
- Add a constraint: Sum of x_2 and x_6 should not be more than 2.
  - Mathematically, $x_2 + x_6 \leq 2$

- Close the file, type: `make` and to run, type: `./sbnb`

More test instances on MINLPLib: A Library of Mixed-Integer and Continuous Nonlinear

Problem instance generation and solving

Consider the problem

$$\begin{aligned} Min\ & x_0^{-1} + 2x_1^{-2} + 3x_2^{-1.5} + 4x_3^{-1.7} + 5x_4^{-1.2} + 6x_5^{-1.7} + 7x_6^{-1.4} + 8x_7^{-1.2} + 9x_8^{-1.5} \\ s.t.\ & x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 \leq 64 \\ & 1 \leq x_i \leq 64,\ \text{integer}\ \ \forall i = 0, \ldots, 8. \end{aligned}$$

Let us try!

- Type: `cd ~/minotaur/examples/orsi2018`
- Open the file `SimpleBnb.cpp` (**type:** `gedit SimpleBnb.cpp`)
- Go to the main function
  - function to create problem
  - solve function
- Close the file, type: `make` and to run, type: `./sbnb`
- Add a constraint: Sum of x_2 and x_6 should not be more than 2.
  - Mathematically, $x_2 + x_6 \leq 2$
  - **In `SimpleBnb.cpp` in `createProblem` function**

```
LinearFunctionPtr lf1 = (LinearFunctionPtr) new LinearFunction();
FunctionPtr fun1;
lf1->addTerm(vars[2],1.0);
lf1->addTerm(vars[6],1.0);
fun1 = (FunctionPtr) new Function(lf1);
p->newConstraint(fun1, -INFINITY, 2);
```

- Close the file, type: `make` and to run, type: `./sbnb`

More test instances on MINLPLib: A Library of Mixed-Integer and Continuous Nonlinear

Customize an existing brancher

Different branching rules:

- Lexicographic: choose *candidate* with smallest index
- Maximum violation: choose the most fractional candidate
  - $x_1 = 0.9,\ \text{score} = 0.1(0.8) + 0.9 * (0.2) = 0.26$
  - $x_5 = 0.2,\ \text{score} = 0.2(0.8) + 0.8 * (0.2) = 0.32$
  - $x_6 = 0.4,\ \text{score} = 0.4(0.8) + 0.6 * (0.2) = 0.44$

Branching rule: Perform minimum violation based selection

Let us try

- Open the file `MinVioBrancher.cpp`
- Locate the function `findBestCandidate_()`
- Make the required changes as discussed
- Close the file and open `SimpleBnb.cpp`
  - include the brancher to the file
  - use this brancher
- Close the file, type: `make` and to run, type: `./sbnb`

A new brancher MaxVioBinFirst

Branching rule: Select the Maximum Violation Binary Variable First

Same as MaxVio but in addition gives preference to binary variables

Let us try

- To open the file **type:** `MaxVioBinFirstBrancher.cpp`
- Locate the function `findBestCandidate_()`
- Observe the implementation of the branching rule
- Close the file and **type:** `make maxVioBinFirst`
- To run **type:** `./bfbnb instances/tls2.nl > tls2.out`
- See the output in `tls2.out`
- observe:
  - branching candidates and best branching candidate at different nodes
  - solve statistics

Play around with instances in the library minlpLib

THANK YOU.

For any discussions/questions, please contact:

- Prof. Ashutosh Mahajan (amahajan@iitb.ac.in)
- Prashant Palkar (prashant.palkar@iitb.ac.in)
- Meenarli Sharma (meenarli@iitb.ac.in)