Outline

1. Problem Definition and Assumptions
2. Nonlinear Branch-and-Bound
3. Advanced Nonlinear Branch-and-Bound
Mixed-Integer Nonlinear Optimization

Mixed-Integer Nonlinear Program (MINLP)

\[
\begin{align*}
\text{minimize} \quad & f(x) \\
\text{subject to} \quad & c(x) \leq 0 \\
& x \in \mathcal{X} \\
& x_i \in \mathbb{Z} \text{ for all } i \in \mathcal{I}
\end{align*}
\]

Basic Assumptions for Convex MINLP

**A1** \( \mathcal{X} \) is a bounded polyhedral set.

**A2** \( f \) and \( c \) twice continuously differentiable convex

**A3** MINLP satisfies a constraint qualification.

**A2** (convexity) most restrictive (show how to relax later)

**A3** is technical (MFCQ would have been sufficient)
Overview of Basic Methods

Two broad classes of method

1. Single-tree methods; e.g.
   - Nonlinear branch-and-bound
   - LP/NLP-based branch-and-bound
   - Nonlinear branch-and-cut
   ... build and search a single tree

2. Multi-tree methods; e.g.
   - Outer approximation
   - Benders decomposition
   - Extended cutting plane method
   ... alternate between NLP and MILP solves

Multi-tree methods only evaluate functions at integer points

Concentrate on methods for convex problems today.

Can mix different methods & techniques.
Outline

1. Problem Definition and Assumptions
2. Nonlinear Branch-and-Bound
3. Advanced Nonlinear Branch-and-Bound
Nonlinear Branch-and-Bound

Solve NLP relaxation ($x_i$ continuous, not integer)

\[
\minimize_x f(x) \text{ subject to } c(x) \leq 0, \quad x \in \mathcal{X}
\]

- If $x_i \in \mathbb{Z} \ \forall \ i \in \mathcal{I}$, then solved MINLP
- If relaxation is infeasible, then MINLP infeasible

... otherwise search tree whose nodes are NLPs:

\[
\begin{align*}
\minimize_x f(x), \\
\text{subject to } c(x) \leq 0, \\
x \in \mathcal{X}, \\
\quad l_i \leq x_i \leq u_i, \quad \forall i \in \mathcal{I}.
\end{align*}
\]

NLP relaxation is NLP($-\infty, \infty$) ... search tree
Nonlinear Branch-and-Bound

Solve relaxed NLP \((0 \leq x_I \leq 1\) continuous relaxation) … solution value provides lower bound

- Branch on \(x_i\) non-integral
- Solve NLPs & branch until
  1. Node infeasible: \(\bullet\)
  2. Node integer feasible: \(\square\) \(\Rightarrow\) get upper bound \((U)\)
  3. Lower bound \(\geq U\): \(\triangleup\)

Search until no unexplored nodes

Software:

- GAMS-SBB, MINLPBB [L]
- BARON [Sahinidis] global
- Couenne [Belotti] global
Nonlinear Branch-and-Bound

Branch-and-bound for MINLP
Choose tol $\epsilon > 0$, set $U = \infty$, add $(\text{NLP}(-\infty, \infty))$ to heap $\mathcal{H}$.

while $\mathcal{H} \neq \emptyset$ do
  Remove $(\text{NLP}(l, u))$ from heap: $\mathcal{H} = \mathcal{H} - \{ \text{NLP}(l, u) \}$.
  Solve $(\text{NLP}(l, u)) \Rightarrow$ solution $x^{(l,u)}$
  if $(\text{NLP}(l, u))$ is infeasible then
    Prune node: infeasible
  else if $f(x^{(l,u)}) > U$ then
    Prune node; dominated by bound $U$
  else if $x^{(l,u)}$ integral then
    Update incumbent: $U = f(x^{(l,u)})$, $x^* = x^{(l,u)}$.
  else
    BranchOnVariable($x_i^{(l,u)}$, $l$, $u$, $\mathcal{H}$)
  end
end
Nonlinear Branch-and-Bound

BnB is finite, provided $\mathcal{X}$ is bounded polyhedron:

**Theorem (Finiteness of Nonlinear Branch-and-Bound)**

Solve MINLP by nonlinear branch-and-bound, and assume that A1-A3 hold. Then BnB terminates at optimal solution (or indication of infeasibility) after a finite number of nodes.

**Proof.**

- (A1-A3) $\Rightarrow$ every NLP solved globally (convex, MFCQ)
- Boundedness of $\mathcal{X}$ $\Rightarrow$ tree is finite
  $\Rightarrow$ convergence, see e.g. Theorem 24.1 of [Schrijver, 1986]. □
Nonlinear Branch-and-Bound

BnB trees can get pretty large ...

Synthesis MINLP B&B Tree: 10000+ nodes after 360s

... be smart about solving NLPs & searching tree!
Outline

1 Problem Definition and Assumptions

2 Nonlinear Branch-and-Bound

3 Advanced Nonlinear Branch-and-Bound
Advanced Nonlinear BnB

Basic BnB will work, but needs improvements:

- Selection of branching variables
- Node selection strategies
- Inexact NLP solves & hot-starts
- Cutting planes & branch-and-cut
- Software design & modern solvers, e.g. MINOTAUR

... critical for efficient implementation
Advanced Nonlinear BnB: Variable Selection

Ideally choose branching sequence to minimize tree size
... impossible in practice; sequence not known a priori
⇒ choose variable that maximizes increase in lower bound

Let $I_c \subset I$ set of candidates: fractional integer variables
... in practice choose subset of important variables (priorities)

Maximum Fractional Branching
Branch on variable $i_0$ with largest integer violation:

$$i_0 = \arg \max_{i \in I_c} \left\{ \min \left( x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i \right) \right\},$$

... as bad as random branching [Achterberg et al., 2004]
Advanced Nonlinear BnB: Variable Selection

Successful rules estimate change in lower bound after branching

- Increasing lower bound improves pruning
- For $x_i, i \in I$, define degradation estimates $D_i^+$ and $D_i^-$ for increase in lower bound
- Goal: make both $D_i^+$ and $D_i^-$ large!
- Combine $D_i^+$ and $D_i^-$ into single score:

$$s_i := \mu \min(D_i^+, D_i^-) + (1 - \mu) \max(D_i^+, D_i^-),$$

where parameter $\mu \in [0, 1]$ close to 1.

Degradation-Based Branching

Branch on variable $i_0$ with largest degradation estimate:

$$i_0 = \arg \max_{i \in I_c} \{ s_i \}$$

... methods differ by how $D_i^+$ and $D_i^-$ computed
The first approach for computing degradations is ...

**Strong Branching**

Solve $2 \times |\mathcal{I}_c|$ NLPs for every potential child node:

- Solution at current (parent) node ($\text{NLP}(l, u)$) is $f_p := f^{(l, u)}$
- $\forall x_i, i \in \mathcal{I}_c$ create two temporary NLPs: $\text{NLP}_i(l^-, u^-)$ and $\text{NLP}_i(l^+, u^+)$
- Solve both NLPs ...  
  ... if both infeasible, then prune ($\text{NLP}(l, u)$)  
  ... if one infeasible, then fix integer in parent ($\text{NLP}(l, u)$)  
  ... otherwise, let solutions be $f_i^+$ and $f_i^-$ and compute

\[
D_i^+ = f_i^+ - f_p, \quad \text{and} \quad D_i^- = f_i^- - f_p.
\]
Advanced Nonlinear BnB: Variable Selection

Advantage/Disadvantage of strong branching:
- **Good**: Reduce the number of nodes in tree
- **Bad**: Slow overall, because too many NLPs solved
- Solving NLPs approximately does not help

Fact: **MILP ≠ MINLP**

LPs hot-start efficiently (re-use basis factors), but NLPs cannot be warm-started (neither IPM nor SQP)!

Reason (NLPs are, well ... nonlinear):
- NLP methods are iterative: generate sequence \( \{x^{(k)}\} \)
- At solution, \( x^{(l)} \), have factors from \( x^{(l-1)} \) ... out-of-date
Approximate Strong Branching

Simple idea: Use QP (LP) approximation [Bonami et al., 2011]

CPU[s] for root node and round (2 # ints) of strong branching:

<table>
<thead>
<tr>
<th>problem</th>
<th># ints</th>
<th>Full NLP</th>
<th>Cold QP</th>
<th>Hot QP</th>
</tr>
</thead>
<tbody>
<tr>
<td>stockcycle</td>
<td>480</td>
<td>4.08</td>
<td>3.32</td>
<td>0.532</td>
</tr>
<tr>
<td>RSyn0805H</td>
<td>296</td>
<td>78.7</td>
<td>69.8</td>
<td>1.94</td>
</tr>
<tr>
<td>SLay10H</td>
<td>180</td>
<td>18.0</td>
<td>17.8</td>
<td>1.25</td>
</tr>
<tr>
<td>Syn30M03H</td>
<td>180</td>
<td>40.9</td>
<td>14.7</td>
<td>2.12</td>
</tr>
</tbody>
</table>

- Small savings from replacing NLP by QP solves.
- Order of magnitude saving from re-using factors.
Approximate Strong Branching

**Hot-QP Starts in BQPD [Fletcher]**
- parent node is dual feasible after branching
- perform steps of dual active-set method to get primal feasible
- re-use factors of basis $B = LU$
- re-use factors of dense reduced Hessian $Z^T H Z = L^T D L$
- use $L U$ and $L^T D L$ to factorize KKT system

\[
\begin{bmatrix}
H & A^T \\
A & 0
\end{bmatrix}
\]

where $B^{-1} = [A : V]^{-1} = \begin{bmatrix} Y \\ Z \end{bmatrix}$

- 2-3 pivots to re-optimize independent of problem size
Approximate Strong Branching

Parametric QP solve
Performance Profiles [Dolan and More, 2002]

Performance profiles
Clever way to display a benchmark

\[ \forall \text{solver } s \quad \log_2 \left( \frac{\# \text{ iter}(s, p)}{\text{best}_\text{iter}(p)} \right) \]

\[ p \in \text{problem} \]

- “probability distribution”: solver “A” is at most x-times slower than best.
- Origin shows percentage of problems where solver “A” is best.
- Asymptotics shows reliability of solver “A”.
Performance Profiles (Formal Definition)

Performance ratio of $t_{p,s}$ for $p \in \mathcal{P}$ of problems, $s \in S$ of solvers:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S, \}}$$

distribution function $\rho_s(\tau)$ for solver $s \in S$

$$\rho_s(\tau) = \frac{\text{size}\{p \in \mathcal{P} \mid r_{p,s} \leq \tau\}}{|\mathcal{P}|}.$$ 

$\rho_s(\tau)$ probability that solver $s$ is at most $\tau \times$ slower than best
Approximate Strong Branching

Performance (\# nodes) of NLP/QP/LP strong branching
Approximate Strong Branching

Performance (CPU time) of NLP/QP/LP strong branching
Pseudocost Branching

Keep history of past branching to estimate degradations

- \( n_i^{+}, n_i^{-} \) number of times up/down node solved for variable \( i \)
- \( p_i^{+}, p_i^{-} \) pseudocosts updated when child solved:

\[
p_i^{+} = \frac{f_i^{+} - f_p}{\lceil x_i \rceil - x_i} + p_i^{+}, \quad n_i^{+} = n_i^{+} + 1 \quad \text{or} \quad p_i^{-} = \ldots \quad n_i^{-} = \ldots
\]

- Compute estimates of \( D_i^{+} \) and \( D_i^{-} \) or branching:

\[
D_i^{+} = (\lceil x_i \rceil - x_i) \frac{p_i^{+}}{n_i^{+}} \quad \text{and} \quad D_i^{-} = (x_i - \lfloor x_i \rfloor) \frac{p_i^{-}}{n_i^{-}}.
\]

- Initialize pseudocosts with strong branching
- Good estimates for MILP, \cite{LinderothSavelsbergh1999} 
- Not clear how to update, if NLP infeasible ... \( \ell_1 \) penalty?
Advanced Nonlinear BnB: Variable Selection

Following approach combines strong branching and pseudocosts

**Reliability Branching**

- Strong branching early, then pseudocost branching
  - While $n_i^+$ or $n_i^- \leq \tau (= 5)$ do strong branching on $x_i$
  - Once $n_i^+$ or $n_i^- > \tau$ switch to pseudocost

Important alternatives to variables branching:
- SOS branching, see [Beale and Tomlin, 1970]
- Branching on split disjunctions

$$\left( a^T x_I \leq b \right) \lor \left( a^T x_I \geq b + 1 \right)$$

where $a \in \mathbb{Z}^p$ and $b \in \mathbb{Z}$ … conceptually like conjugate directions
Advanced Nonlinear BnB: Node Selection

Strategic decision on which node to solve next.

Goals of node selection
- Find good feasible solution quickly to reduce upper bound, $U$
- Prove optimality of incumbent $x^*$ by increasing lower bound

Popular strategies:
1. Depth-first search
2. Best-bound search
3. Hybrid schemes
Advanced Nonlinear BnB: Depth-First Search

Depth-First Search
Select deepest node in tree (or last node added to heap $\mathcal{H}$)

Advantages:
- Easy to implement (Sven likes that ;-)  
- Keeps list of open nodes, $\mathcal{H}$, as small as possible  
- Minimizes the change to next NLP ($\text{NLP}(l, u)$):  
  ... only single bound changes $\Rightarrow$ better hot-starts

Disadvantages:
- poor performance if no upper bound is found:  
  $\Rightarrow$ explores nodes with a lower bound larger than solution
Advanced Nonlinear BnB: Best-Bound Search

Best-Bound Search

Select node with best lower bound

Advantages:

- Minimizes number of nodes for fixed sequence of branching decisions, because all explored nodes would have been explored independent of upper bound

Disadvantages:

- Requires more memory to store open problems
- Less opportunity for warm-starts of NLPs
- Tends to find integer solutions at the end
Advanced Nonlinear BnB: Best-Bound Search

1. **Best Expected Bound**: node with best bound after branching:

   \[ b_p^+ = f_p + (\lceil x_i \rceil - x_i) \frac{p_i^+}{n_i^+} \quad \text{and} \quad b_p^- = f_p + (x_i - \lfloor x_i \rfloor) \frac{p_i^-}{n_i^-}. \]

   Next node is \( \max_p \left\{ \min (b_p^+, b_p^-) \right\} \).

2. **Best Estimate**: node with best expected solution in subtree

   \[ e_p = f_p + \sum_{i: x_i \text{fractional}} \min \left( (\lceil x_i \rceil - x_i) \frac{p_i^+}{n_i^+}, (x_i - \lfloor x_i \rfloor) \frac{p_i^-}{n_i^-} \right), \]

   Next node is \( \max_p \{ e_p \} \).

... good search strategies combine depth-first and best-bound
Advanced Nonlinear BnB: Inexact NLP Solves

Role for inexact solves in MINLP

- Provide approximate values for strong branching
- Solve NLPs inexactly during tree-search:
  - [Borchers and Mitchell, 1994] consider single SQP iteration
    ... perform early branching if limit seems non-integral
    ... augmented Lagrangian dual for bounds
  - [Leyffer, 2001] considers single SQP iteration
    ... use outer approximation instead of dual
    ... numerical results disappointing

... reduce solve time by factor 2-3 at best

- New idea: search QP tree & exploit hot-starts for QPs
  ... QP-diving discussed next ...
Advanced Nonlinear BnB: QP-Diving

Branch-and-bound solves huge number of NLPs ⇒ bottleneck!

QP-Diving Tree-Search:
- solve root node & save factors from last QP solve
- same KKT for whole subtree
- perform MIQP tree-searches
  - depth-first search: ⇒ fast hot-starts
  - back-track: warm-starts

Need new fathoming rules ...

... alternative: change QP approximation after back-track
Advanced Nonlinear BnB: QP-Diving

Assume MINLP is convex

QP-Diving Tree-Search:
Solve QPs until

1. QP infeasible: \( \bullet \)
   ... QP is relaxation of NLP

2. Node integer feasible: \( \square \)
   \( \Rightarrow \) NLP to get upper bnd (\( U \))
   ... QP over-/under-estimates
   \( \Rightarrow \) resolve

3. Infeasible O-cut \( \eta < U \): \( \triangle \)
   Linear O-cut: \( \eta \geq f_k + g_k^T d \)

\( x_i = 0 \) \hspace{1cm} \( x_i = 1 \)
New Extended Performance Profiles

Performance ratio of $t_{p,s}$ for $p \in P$ of problems, $s \in S$ of solvers:

$$\hat{r}_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S, i \neq s\}}$$

distribution function $\rho_s(\tau)$ for solver $s \in S$

$$\hat{\rho}_s(\tau) = \frac{\text{size}\{p \in P \mid \hat{r}_{p,s} \leq \tau\}}{|P|}.$$

- $\hat{\rho}_s(\tau)$ probability that solver $s$ is at most $\tau \times$ slower than best
- For $\hat{r}_{p,s} \geq 1$ get standard performance profile
- Extension: $\hat{r}_{p,s} < 1$ if solver $s$ is fastest for instance $p$
- $\hat{\rho}_s(0.25)$ probability that solver $s$ is $4 \times$ faster than others
CPU-Times for MINOTAUR with Hot-Starts (IPOPT)

Hot-started QP give a huge improvement
CPU-Times for MINOTAUR with Hot-Starts (filterSQP)

Hot-started QP give a huge improvement
Typical Results

<table>
<thead>
<tr>
<th>Solver</th>
<th>CPU</th>
<th>NLPs</th>
<th>CPU/100NLPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPOPT</td>
<td>7184.91</td>
<td>69530</td>
<td>10.3335</td>
</tr>
<tr>
<td>filterSQP</td>
<td>7192.54</td>
<td>37799</td>
<td>19.0284</td>
</tr>
<tr>
<td>QP-Diving</td>
<td>5276.23</td>
<td>1387837</td>
<td>0.3802</td>
</tr>
</tbody>
</table>

⇒ many more nodes ... a little faster.

<table>
<thead>
<tr>
<th>Solver</th>
<th>CPU</th>
<th>NLPs</th>
<th>CPU/100NLPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPOPT</td>
<td>1951.1</td>
<td>16486</td>
<td>11.8349</td>
</tr>
<tr>
<td>filterSQP</td>
<td>849.74</td>
<td>16717</td>
<td>5.0831</td>
</tr>
<tr>
<td>QP-Diving</td>
<td>97.89</td>
<td>24029</td>
<td>0.4074</td>
</tr>
</tbody>
</table>

⇒ similar number of nodes ... much faster!
MINOTAUR: A New Software Framework for MINLP

Mixed Integer Nonlinear Optimization Toolkit: Algorithms, Underestimators & relaxations

Goal: Implement a Range of Algorithms in Common Framework

- Fast, usable MINLP solver.
- Flexibility for developing new algorithms.
- Ease of developing new algorithms.
MINOTAUR’s Four Main Components

**Interfaces** for reading input
- AMPL

**Engines** to solve LP/NLP/QP
- QP: BQPD
- NLP: FilterSQP/IPOPT
- LP: OSI-CLP

**Algorithms** to solve MINLP
- Branch-and-Bound
- Outer-Approximation
- Quesada-Grossmann
- Branch-and-Refine

**Base**
- Data Structures:
  - Problem
  - Objective & Constraints
  - Functions
  - Modifications
  - Gradient, Jacobian, Hessian
- Tools for Search:
  - Node Processors
  - Node Relaxers
  - Branchers
  - Tree Manager
- Utilities
  - Loggers & Timers
  - Options
MINOTAUR’s Four Main Components

*Interfaces* for reading input
- AMPL
- Your Interface Here

*Engines* to solve LP/NLP/QP
- QP: BQPD
- NLP: FilterSQP/IPOPT
- LP: OSI-CLP
- Your engine here

*Algorithms* to solve MINLP
- Branch-and-Bound
- Outer-Approximation
- Quesada-Grossmann
- Branch-and-Refine
- Your algorithm here

*Base*

- Your Data Structures:
  - Problem
  - Objective & Constraints
  - Functions
  - Modifications
  - Gradient, Jacobian, Hessian

- Your Tools for Search:
  - Node Processors
  - Node Relaxers
  - Branchers
  - Tree Manager

- Utilities
  - Loggers & Timers
  - Options

Highly Customizable
## Features

<table>
<thead>
<tr>
<th></th>
<th>Bonmin</th>
<th>FilMINT</th>
<th>BARON</th>
<th>Couenne</th>
<th>Minotaur</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithms:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLP B&amp;B</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Branch &amp; Cut</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Branch &amp; Reduce</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Support for Nonlinear Functions:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comput. Graph</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓ ✓</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>Nonlin. Reform.</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Native Derivat.</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>× ✓</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td><strong>Interfaces:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIMMS</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>× ✓</td>
</tr>
<tr>
<td>AMPL</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
</tr>
<tr>
<td>GAMS</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓ ✓ ✓</td>
<td>✓</td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>× ✓</td>
<td>✓ ✓</td>
</tr>
</tbody>
</table>
MINOTAUR Performance

Time taken for 463 MINLP Instances from GAMS, MacMINLP, CMU test-sets.
MINOTAUR’s Soft-Wear Stack

... available at www.mcs.anl.gov/minotaur
Summary and Teaching Points

Nonlinear Branch-and-Bound

- Solves problem by branching globally for convex MINLPs
- Need careful implementation of
  - Branching variable & node selection
  - Software infrastructure ... build on other frameworks!
  - Interfaces to NLP and other solvers
- Exploit linearity (or QP) as much as possible
  - QP diving works really well for many MINLPs
  - Approximate tree-search reduces CPU time
- Implementation matters ... many open-source solvers


An outer-inner approximation for separable MINLPs.
Technical report, LIF, Faculté des Sciences de Luminy, Université de Marseille.


