Mixed-Integer Nonlinear Optimization: Algorithms for Convex Problems

GIAN Short Course on Optimization: Applications, Algorithms, and Computation

Sven Leyffer

Argonne National Laboratory

September 12-24, 2016

Outline

Mixed-Integer Nonlinear Optimization

Mixed-Integer Nonlinear Program (MINLP)

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

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

Nonlinear Branch-and-Bound

Solve NLP relaxation ($x_{\mathcal{I}}$ continuous, not integer)

$$\underset{x}{\text{minimize}} \ f(x) \quad \text{subject to } c(x) \leq 0, \ x \in \mathcal{X}$$

- 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:

$$\begin{cases} \underset{x}{\text{minimize}} & f(x), \\ \text{subject to} & c(x) \leq 0, \\ & x \in \mathcal{X}, \\ & l_i \leq x_i \leq u_i, \ \forall i \in \mathcal{I}. \end{cases} \qquad (\text{NLP}(l, u))$$

NLP relaxation is $\text{NLP}(-\infty, \infty)$ ... search tree

Nonlinear Branch-and-Bound

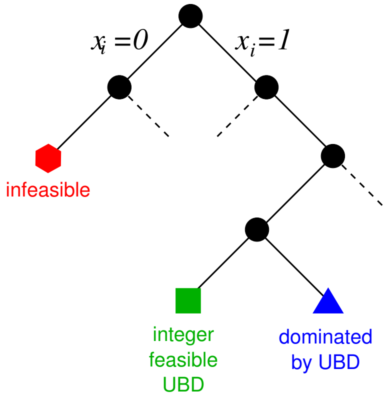
Solve relaxed NLP ($0 \leq x_{\mathcal{I}} \leq 1$ continuous relaxation)
. . . solution value provides lower bound

- Branch on x_i non-integral
- Solve NLPs & branch until
  1. Node infeasible: ●
  2. Node integer feasible: □
     ⇒ get upper bound (U)
  3. Lower bound $\geq U$: ▲

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 (NLP$(-\infty, \infty)$) to heap $\mathcal{H}$.

while $\mathcal{H} \neq \emptyset$ **do**

  Remove (NLP(l, u)) from heap: $\mathcal{H} = \mathcal{H} - \{$ NLP(l, u) $\}$.

  Solve (NLP(l, u)) $\Rightarrow$ solution $x^{(l,u)}$

  if *(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 $x_{\mathcal{I}}^{(l,u)}$ *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.

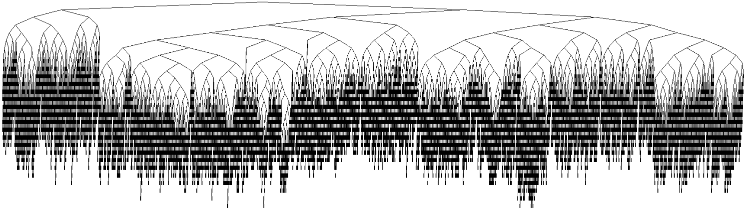
Proof.

- (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]. $\square$

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

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
$\Rightarrow$ choose variable that maximizes increase in lower bound

Let $\mathcal{I}_c \subset \mathcal{I}$ 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:

$$i_0 = \underset{i \in \mathcal{I}_c}{\operatorname{argmax}} \left\{ \min \left(x_i - \lfloor x_i \rfloor \, , \, \lceil x_i \rceil - x_i \right) \right\} ,$$

... 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 $x_i, i \in \mathcal{I}$, 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:

$$i_0 = \underset{i \in \mathcal{I}_c}{\operatorname{argmax}} \{s_i\}$$

... methods differ by how D_i^+ and D_i^- computed

Advanced Nonlinear BnB: Variable Selection

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 (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 (NLP(l, u))
  ... if one infeasible, then fix integer in parent (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 $\neq$ 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:

problem	# ints	Full NLP	Cold QP	Hot QP
stockcycle	480	4.08	3.32	0.532
RSyn0805H	296	78.7	69.8	1.94
SLay10H	180	18.0	17.8	1.25
Syn30M03H	180	40.9	14.7	2.12

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

Approximate Strong Branching

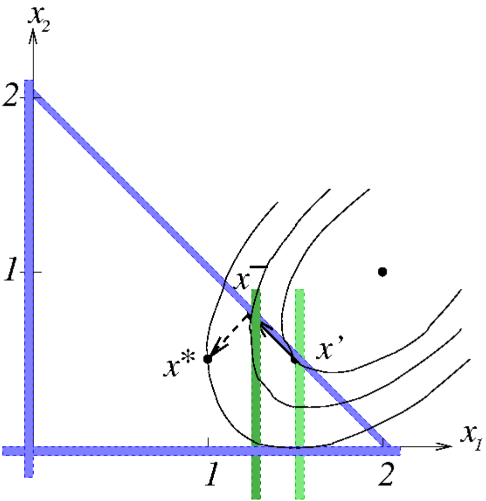
Hot-QP Starts in BQPD [Fletcher]

- 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 $Z^T HZ = L^T DL$
- use LU and $L^T DL$ to factorize KKT system

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \quad \text{where} \quad B^{-1} = [A : V]^{-1} = \begin{bmatrix} Y \\ Z \end{bmatrix}$$

- 2-3 pivots to re-optimize independent of problem size

Approximate Strong Branching



Parametric QP solve

Performance Profiles [Dolan and More, 2002]

name	Random CPU	Random nodes	Most-Fractional CPU	Most-Fractional nodes
BatchS101006M	141.9	9464	68.7	7560
BatchS121208M	2694.8	96566	566.1	41600
BatchS151208M	6781.6	176188	1710.0	102744
BatchS201210M	> 10800	> 174400	6050.6	275740
CLay0204H	61.0	4272	27.3	3404
CLay0204M	7.0	4563	0.7	1361
CLay0205H	271.9	[illegible]	[illegible]	81922
CLay0205M	338.1	[illegible]	[illegible]	22695
CLay0303M	[illegible]	[illegible]	[illegible]	1032
CLay0304H	[illegible]	[illegible]	[illegible]	11631
CLay0304M	[illegible]	[illegible]	[illegible]	30698
CLay0305H	[illegible]	160483	2169.1	70552
CLay0305M	710.9	185254	56.7	38282
FLay04H	49.3	3158	37.1	3012
FLay04M	1.9	3294	1.4	2504
FLay05H	8605.9	185954	4781.3	129598
FLay05M	215.2	188346	125.3	114122
FLay06M	> 10800	> 5166800	> 10800	> 6022600

No!!!

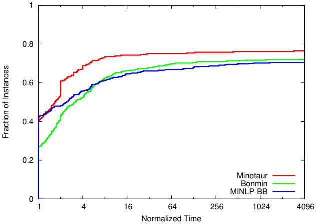
Performance profiles

Clever way to display a benchmark

$$\forall \ \text{solver} \ \ s \qquad \log_2\left(\frac{\#\ \text{iter}(s,p)}{\text{best\_iter}(p)}\right)$$

$$p \in \text{problem}$$

- "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:

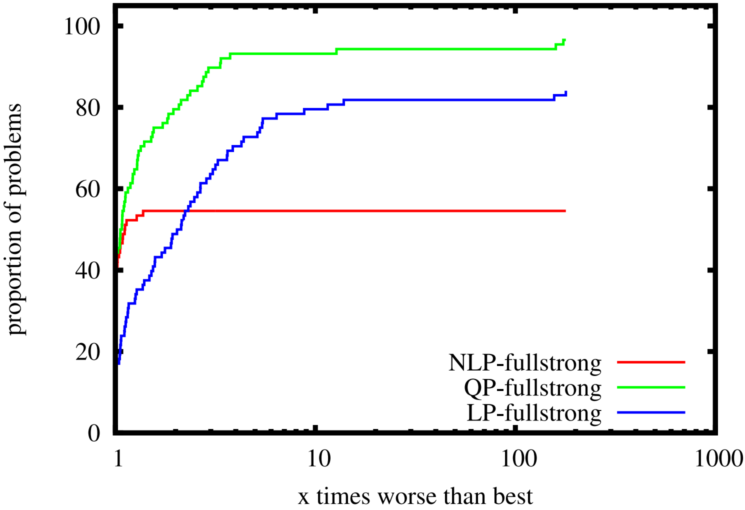
$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S, \}}$$

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

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

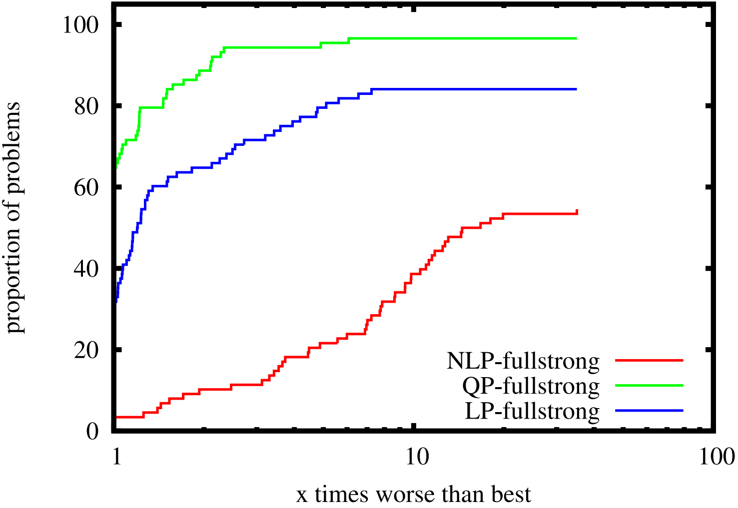
$\rho_s(\tau)$ probability that solver s is at most $\tau \times$ slower than best

Approximate Strong Branching



Performance (# nodes) of NLP/QP/LP strong branching

Approximate Strong Branching



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

Advanced Nonlinear BnB: Variable Selection

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:

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

- Compute estimates of D_i^+ and D_i^- or branching:

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

- Initialize pseudocosts with strong branching
- Good estimates for MILP, [Linderoth and Savelsbergh, 1999]
- Not clear how to update, if NLP infeasible ... ℓ_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

$$\left(a^T x_{\mathcal{I}} \leq b\right) \vee \left(a^T x_{\mathcal{I}} \geq b + 1\right)$$

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 (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 $\max_p \left\{ \min\left(b_p^+, b_p^-\right) \right\}$.

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 ...

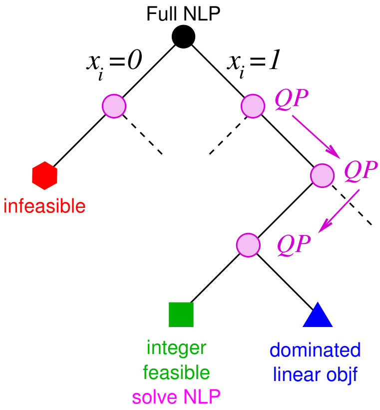
Advanced Nonlinear BnB: QP-Diving

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 ...



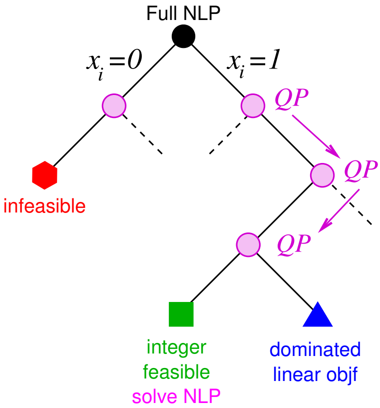
... alternative: change QP approximation after back-track

Advanced Nonlinear BnB: QP-Diving

Assume MINLP is convex

QP-Diving Tree-Search:
Solve QPs until

1. QP infeasible: ●
   ... QP is relaxation of NLP
2. Node integer feasible: □
   $\Rightarrow$ NLP to get upper bnd (U)
   ... QP over-/under-estimates
   $\Rightarrow$ resolve
3. Infeasible O-cut $\eta < U$: ▲
   Linear O-cut: $\eta \geq f_k + g_k^T d$



New Extended Performance Profiles

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

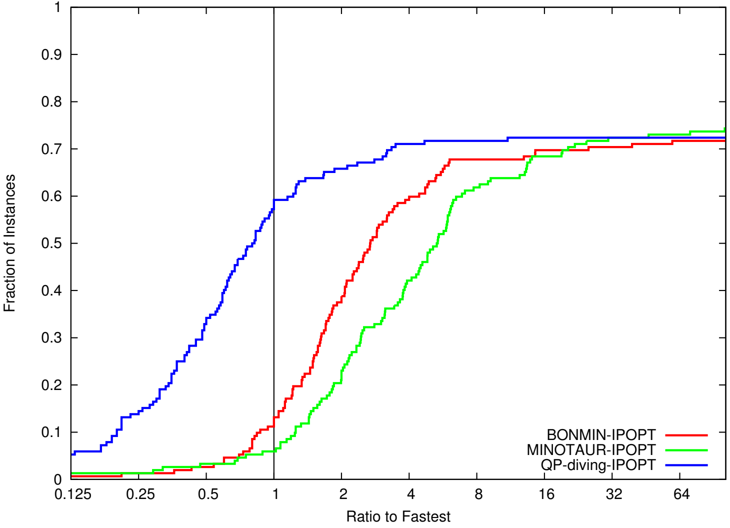
$$\hat{r}_{p,s} = \frac{t_{p,s}}{\min\{t_{p,i} \mid i \in S, i \neq s\}}$$

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

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

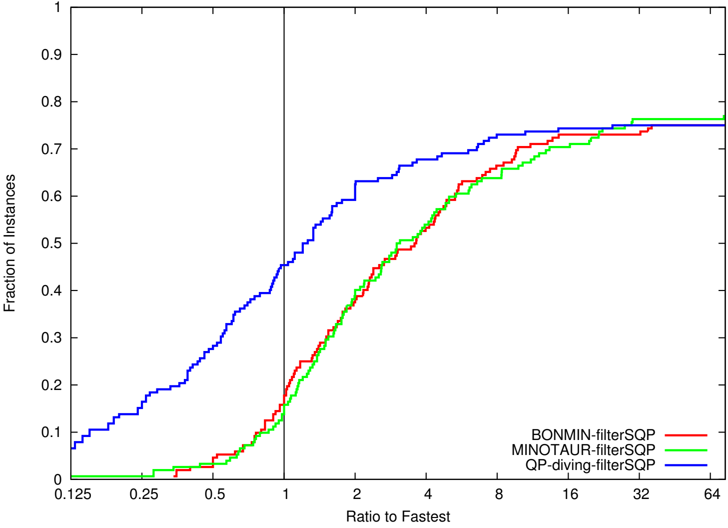
- $\hat{\rho}_s(\tau)$ probability that solver s is at most $\tau\times$ slower than best
- For $\hat{r}_{p,s} \geq 1$ get standard performance profile
- Extension: $\hat{r}_{p,s} < 1$ if solver s is fastest for instance p
- $\hat{\rho}_s(0.25)$ probability that solver s is $4\times$ faster than others

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

RSyn0840M02M

Solver	CPU	NLPs	CPU/100NLPs
IPOPT	7184.91	69530	10.3335
filterSQP	7192.54	37799	19.0284
QP-Diving	5276.23	1387837	0.3802

$\Rightarrow$ many more nodes ... a little faster.

CLay0305H

Solver	CPU	NLPs	CPU/100NLPs
IPOPT	1951.1	16486	11.8349
filterSQP	849.74	16717	5.0831
QP-Diving	97.89	24029	0.4074

$\Rightarrow$ similar number of nodes ... much faster!

MINOTAUR: A New Software Framework for MINLP

Mixed
Integer
Nonlinear
Optimization
Toolkit:
Algorithms,
Underestimators &
Relaxations

It's only half bull

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

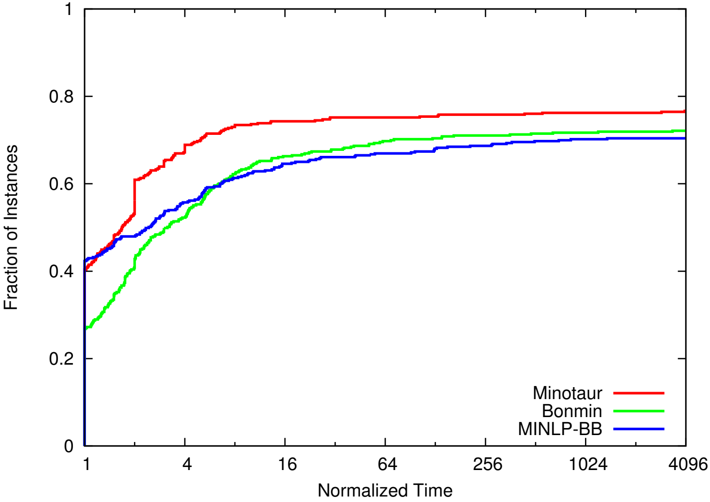
- Your Data Structures:
  - Problem
  - Objective & Constraints
  - Functions
  - Modifications
  - Gradient, Jacobian, Hessian
- Your Tools for Search:
  - Node Processors
  - Node Relaxers
  - Branchers
  - Tree Manager
- Utilities
  - Loggers & Timers
  - Options

Highly Customizable

Features

	Bonmin	FilMINT	BARON	Couenne	Minotaur
Algorithms:					
NLP B&B	✓	×	×	×	✓
Branch & Cut	✓	✓	×	×	✓
Branch & Reduce	×	×	✓	✓	✓
Support for Nonlinear Functions:					
Comput. Graph	×	×	✓	✓	✓
Nonlin. Reform.	×	×	×	×	✓
Native Derivat.	×	×	×	×	✓
Interfaces:					
AIMMS	×	×	✓	×	×
AMPL	✓	✓	×	✓	✓
GAMS	✓	×	✓	✓	×
Open Source	✓	×	×	✓	✓

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

Achterberg, T., Koch, T., and Martin, A. (2004).
Branching rules revisited.
Operations Research Letters, 33:42–54.

Beale, E. and Tomlin, J. (1970).
Special facilities in a general mathematical programming system for non- convex problems using ordered sets of variables.
In Lawrence, J., editor, *Proceedings of the 5th International Conference on Operations Research*, pages 447–454, Venice, Italy.

Bonami, P., Lee, J., Leyffer, S., and Wächter, A. (2011).
More branch-and-bound experiments in convex nonlinear integer programming.
Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division.

Borchers, B. and Mitchell, J. E. (1994).
An improved branch and bound algorithm for mixed integer nonlinear programs.
Computers & Operations Research, 21:359–368.

Duran, M. A. and Grossmann, I. (1986).
An outer-approximation algorithm for a class of mixed-integer nonlinear programs.
Mathematical Programming, 36:307–339.

Geoffrion, A. M. (1972).
Generalized Benders decomposition.
Journal of Optimization Theory and Applications, 10(4):237–260.

Hijazi, H., Bonami, P., and Ouorou, A. (2010).

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

Leyffer, S. (2001).
Integrating SQP and branch-and-bound for mixed integer nonlinear programming.
Computational Optimization & Applications, 18:295–309.

Linderoth, J. T. and Savelsbergh, M. W. P. (1999).
A computational study of search strategies in mixed integer programming.
INFORMS Journal on Computing, 11:173–187.

Schrijver, A. (1986).
Theory of Linear and Integer Programming.
Wiley, New York.

Westerlund, T. and Pettersson, F. (1995).
A cutting plane method for solving convex MINLP problems.
Computers & Chemical Engineering, 19:s131–s136.