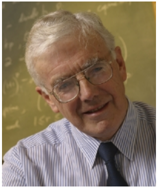
Software for Integer and Nonlinear Optimization

Sven Leyffer, leyffer@mcs.anl.gov
Mathematics & Computer Science Division
Argonne National Laboratory



Roger Fletcher & Jeff Linderoth

Advanced Methods and Perspectives in Nonlinear Optimization and Control
Toulouse, February 3–5, 2010

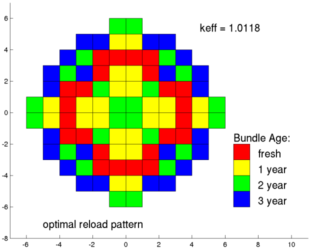
Overview

1. Integer Nonlinear Optimization Applications
   - Problem Statement & Applications
   - DOE Grand-Challenge Application
2. Overview of Existing MINLP Methods
   - Branch-and-Bound
   - Outer Approximation
   - Branch-and-Cut
3. NLP Solver for Branch-and-Bound
   - Motivation
   - Better Branching for MINLP
   - Faster NLPs for MINLP

MINLP Problem & Applications

Mixed Integer Nonlinear Program (MINLP)

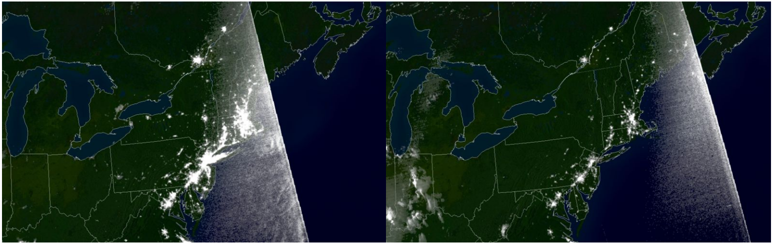
$$
\begin{aligned}
\underset{x,y}{\text{minimize}} \quad & f(x, y) \\
\text{subject to} \quad & c(x, y) \le 0 \\
& y \in Y \text{ integer}
\end{aligned}
$$



Applications:

- distillation column design
- radiation therapy treatment planning
- network design under interdiction
- reactor core reload operation
- gas transmission network design
- optimal discrete control [Sager]

Blackout Prevention in National Power Grid



2003 blackout: before and during

- 2003 blackout cost $4-10 billion and affected 50 million people
- prevent using contingency analysis
  - find least number of lines whose removal results in failure
  - binary variables model removal of lines
  - nonlinearities model power flow
  - results in large nonconvex integer optimization problem
- current analysis limited to 10s of lines

Properties of Electricity Grid Models

Efficient management of electric grid

- optimal power flow problem in network
- predict effects of network expansion/failure

Variables (complex for alternating current)

- voltage: difference in electric potential between nodes
- current: flow of electricity
- power: quantity of energy transferred
  $S = P + iQ$ real/reactive power use polar coordinates

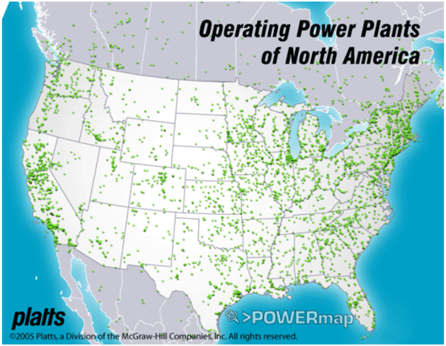
Expression for real power (reactive power similar):

$$P_{ij} = \nu_i^2 \; (y_{ij}\cos(\zeta_{ij}) + g_{ij}) - \nu_i \nu_j y_{ij} \cos(\zeta_{ij} + \theta_i - \theta_j)$$

$\Rightarrow$ nonconvex mixed-integer nonlinear program (MINLP)

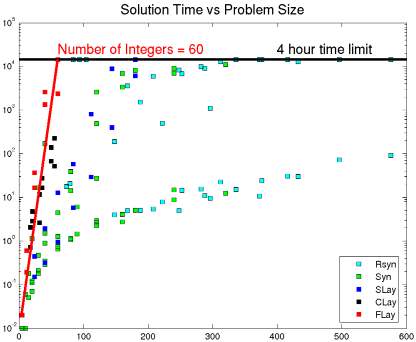
Challenges of Electricity Grid Models

1. Nonconvexities
    - no valid bounds
    - inconsistent subproblems
2. Size
    - 10^5 lines nodes today
    - wind/solar adds order of magnitude
3. Uncertainty
    - supply: goal 20% wind/solar
    - demand: plug-in cars
    - supply & demand: zero-energy buildings



The Curse of Exponentiality

Integer optimization has exponential complexity growth



Time vs number of integers

Parallel MINLP

- 100s of processors get 80% efficiency
- 100,000 processors ... research issues
- perfect speed-up only doubles problem size

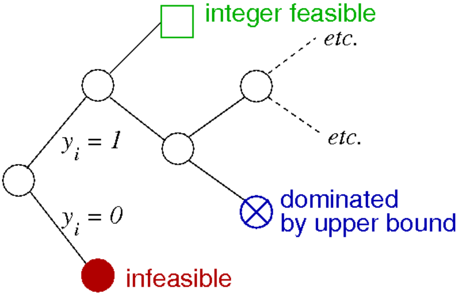
Parallel computing alone not enough: need new methods!

Overview

MINLP Branch-and-Bound [Dakin, 1965]

Solve relaxed NLP ($0 \leq y \leq 1$ continuous relaxation)
...solution value provides lower bound

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

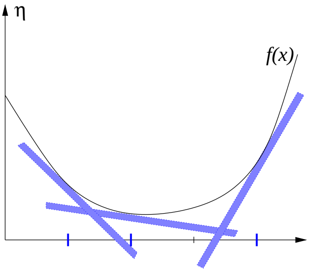


Search until no unexplored nodes on tree

Outer Approximation [Duran & Grossmann, 1986]

NLP subproblem y_j fixed:

$$\text{NLP}(y_j) \begin{cases} \min\limits_{x} & f(x, y_j) \\ \text{s.t.} & c(x, y_j) \le 0 \\ & x \in X \end{cases}$$



linearize f, c about $(x_j, y_j) =: z_j$

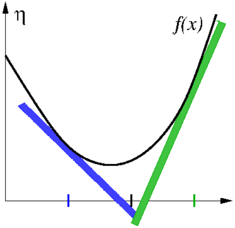
$\Rightarrow$ MINLP $(P) \equiv$ MILP (M)

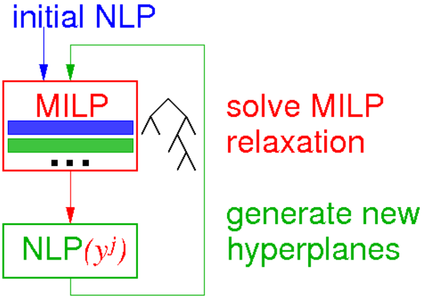
$$(M) \begin{cases} \underset{z=(x,y),\eta}{\text{minimize}} & \eta \\ \text{subject to} & \eta \ge f_j + \nabla f_j^T (z - z_j) \quad \forall y_j \in Y \\ & 0 \ge c_j + \nabla c_j^T (z - z_j) \quad \forall y_j \in Y \\ & x \in X,\ y \in Y \text{ integer} \end{cases}$$

but need linearizations $\forall y_j \Rightarrow$ solve relaxations of (M)

Outer Approximation [Duran & Grossmann, 1986]

Alternate between solve NLP(y_j) and MILP relaxation





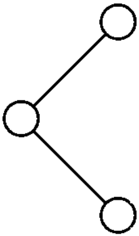
MILP $\Rightarrow$ lower bound; NLP $\Rightarrow$ upper bound

... MILP solution is bottleneck ... no hot-starts for MILP

LP/NLP-Based Branch-and-Bound

AIM: avoid re-solving MILP master (M)

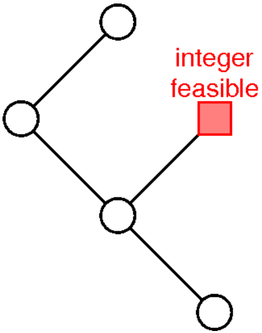
- Take initial MILP tree



LP/NLP-Based Branch-and-Bound

AIM: avoid re-solving MILP master (M)

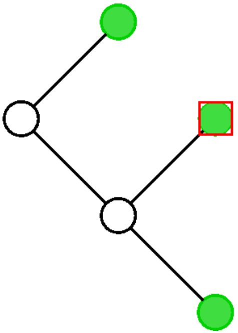
- Take initial MILP tree
- interrupt MILP, when y_j found
  $\Rightarrow$ solve NLP(y_j) get x_j



LP/NLP-Based Branch-and-Bound

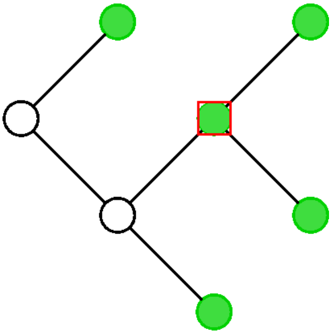
AIM: avoid re-solving MILP master (M)

- Take initial MILP tree
- interrupt MILP, when y_j found
  $\Rightarrow$ solve NLP(y_j) get x_j
- linearize f, c about (x_j, y_j)
  $\Rightarrow$ add linearization to tree



LP/NLP-Based Branch-and-Bound

AIM: avoid re-solving MILP master (M)



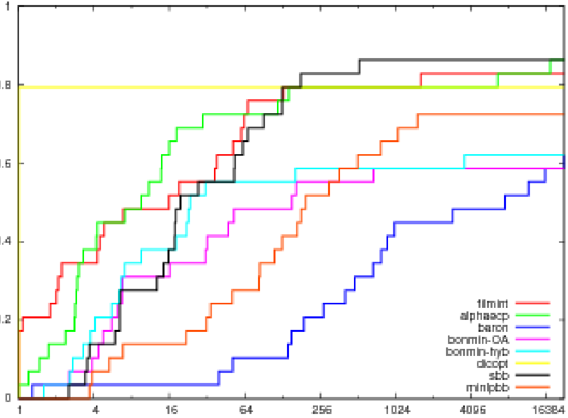
- Take initial MILP tree
- interrupt MILP, when y_j found
  $\Rightarrow$ solve NLP(y_j) get x_j
- linearize f, c about (x_j, y_j)
  $\Rightarrow$ add linearization to tree
- continue MILP tree-search

... until lower bound $\geq$ upper bound

Existing Solvers for MINLP

1. Branch-and-Bound:
   - `GAMS-SBB`, `MINLPBB` standard branch-and-bound
   - `BARON` [Sahinidis], `Couenne` [Belotti] global nonconvex MINLPs
2. Outer Approximation (CPLEX as MIP solver):
   - `DICOPT++` [Grossmann] some heuristics for nonconvex
   - `ECP` [Westerlund] extended cutting plane method
3. LP/NLP-Based Branch-and-Bound:
   - `FilMINT`: FilterSQP + MINTO [Linderoth et al]
   - `BONMIN` [IBM/CMU]
4. Lessons from `FilMINT` & `BONMIN`
   - advanced MILP techniques (`MINTO`, `CBC`): cuts, heuristics
   - aggressive cut (linearization) generation $\simeq$ Kelley's CP method
   - cut management (remove inactive constraints from LPs)

Performance Profile of MINLP Solvers



Probability that solver s at most 2^x times worse than best

Goal: Improve MINLPBB & get it closer to the top!

Exploiting NLP Warm-Starts for MINLP

Motivation:

- CPLEX: improvement in LP important in MIP advances
- Extended Cutting Plane method $10\times$ faster than modern NLP!
- Avoid cutting-plane LPs that grow in size

Opportunities:

1. Better branching decisions: explore alternatives
2. Faster inexact solves vs. lack of bounds

Experiments:

- 56 medium-sized convex MINLPs from IBM/CMU & MacMINLP
- 30-minute time limit (very short for MINLP!)

Maximum-Fractional Branching for MINLP

Goal: Find good integer to branch on

- Change new child nodes as much as possible
- Avoid symmetric trees $\Rightarrow$ identical work

Given solution to parent node

- Find all fractional integers ... denote candidates by y_j, $j \in C$
- Select variable that is closest to 0.5:

$$\max_{j \in C} \quad \{ \lfloor y_j \rfloor + 1 - y_j \ , \ y_j - \lfloor y_j \rfloor \}$$

... only marginally better than random branching!

Pseudo-Cost Branching [Gupta and Ravindran, 1985]

f^p = parent optimum

- solve a child & get f_j^+ or f_j^-
- average pseudo-cost of y_j during tree-search:

$$\mathrm{pc}_j^+ = \frac{f_j^+ - f^p}{\lfloor y_j \rfloor + 1 - y_j} \quad \text{or} \quad \mathrm{pc}_j^- = \frac{f_j^- - f^p}{y_j - \lfloor y_j \rfloor}$$

- compute score$_i$ for all candidates $y_i \in C$:

$$\begin{aligned} = (1-\mu) \quad & \min(\mathrm{pc}_i^-(y_i - \lfloor y_i \rfloor), \mathrm{pc}_i^+(y_i - \lfloor y_i + 1 \rfloor)) \\ +\mu \quad & \max(\mathrm{pc}_i^-(y_i - \lfloor y_i \rfloor), \mathrm{pc}_i^+(y_i - \lfloor y_i + 1 \rfloor)) \end{aligned}$$

- branch on variable i that maximizes score$_i$

initialize pseudo-costs with strong branching ...

Strong Branching for MINLP

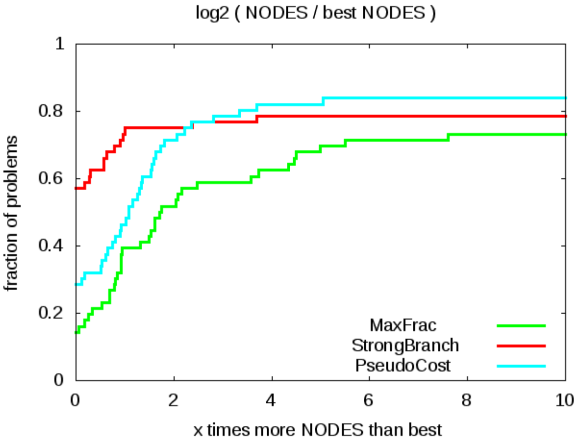
Given solution to parent node NLP, P, with optimum f^p

1. find all non-integral integer variables $y_i, i \in C$
2. for every candidate $y_i \in C$ solve two child NLPs
   - down NLP: $P + \{y_i = \lfloor y_i \rfloor\} \Rightarrow f_i^-$
   - up NLP: $P + \{y_i = \lfloor y_i \rfloor + 1\} \Rightarrow f_i^+$
3. compute score$_i$
   $= (1-\mu) * \min(f_i^- - f^p, f_i^+ - f^p) + \mu * \max(f_i^- - f^p, f_i^+ - f^p)$
4. branch on variable i that maximizes score$_i$ ... where $\mu = 1/6$

Maximize the change in the objective

$\Rightarrow$ variables that changes problem the most [Achterberg et al.]

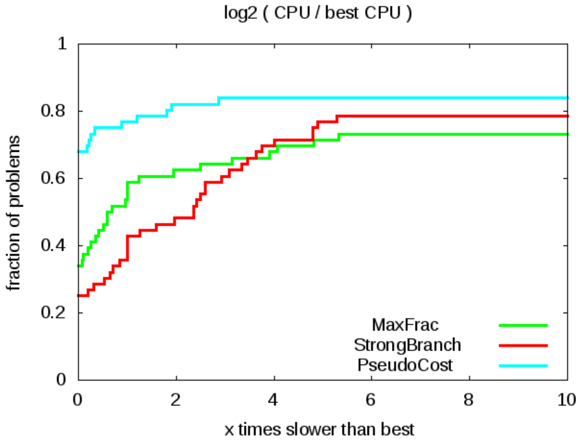
MINLP Performance Profile: Strong-Branching (Nodes)



Number of nodes solved by branching alternatives

Strong branching reduces tree significantly ... less robust in 30 minutes

MINLP Performance Profile: Strong-Branching (CPU)



CPU time for branching alternatives

Jeff: Told ya ... NLP solvers are too slow!

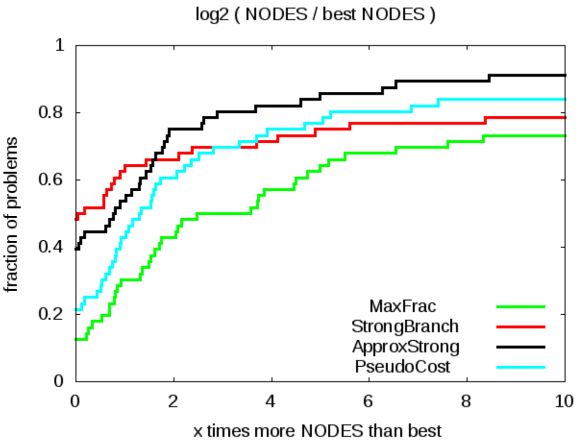
Alternatives to Strong Branching for MINLP

Strong branching: two expensive NLPs per integer per node

Approximate Strong Branching

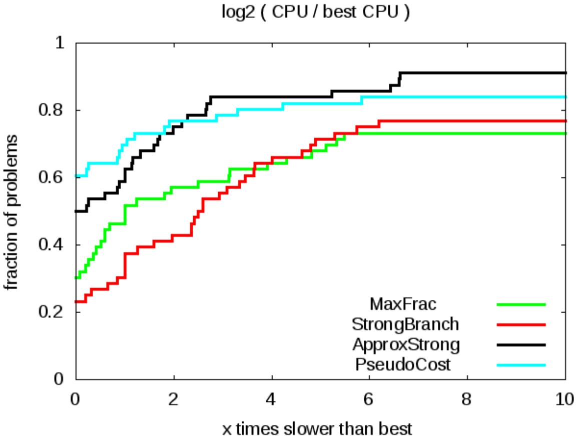
- perform only a few iterations of NLP solver
- easy: set `MaxIter=1` ... single QP solve
- evaluate objective at solution of QP
- single QP estimates nonlinear effect of branching
- no lower bounds from QP: over & underestimate NLPs
  ... depends on nonlinear functions

Performance: Approximate Strong-Branching



Probability that solver *s* at most 2^x more nodes than best

Performance: Approximate Strong-Branching



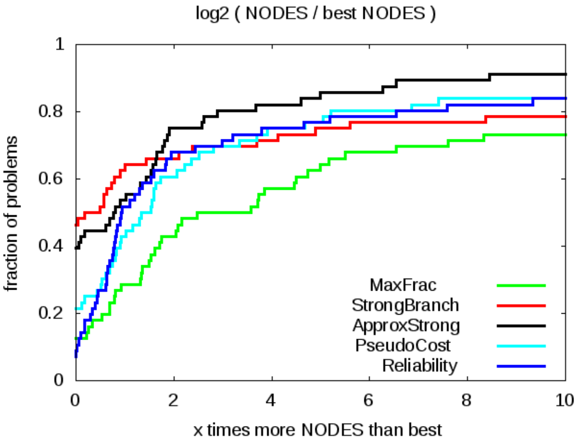
Probability that solver s at most 2^x times slower than best

Alternatives to Strong Branching for MINLP

Reliability Branching

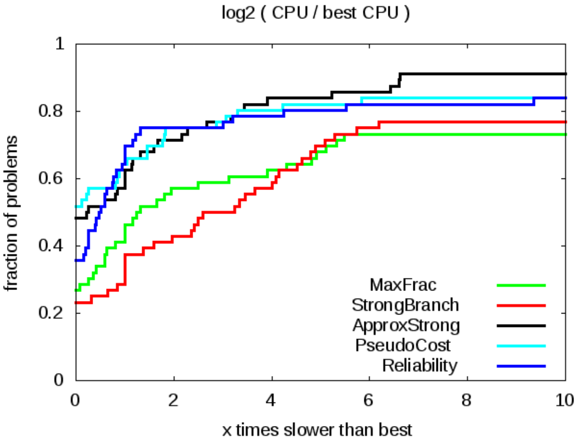
- initialize pseudo-costs with approximate strong branching
- count number of updates of pseudo-cost i: n_{PCi}
- **IF** $n_{PCi} \leq 2$ **THEN**
  score$_i$ from approximate strong branching
  ELSE
  score$_i$ from pseudo-costs
- $n_{PCi} \leq 2$ has not been tuned ...

MINLP Performance Profile (Nodes)



Probability that solver s at most 2^x more nodes than best

MINLP Performance Profile (CPU)



Probability that solver *s* at most 2^x times slower than best

Reliability & ApproxStrong branching work well

Warm-Starts and Hot-Starts for **NLP $\neq$ LP**

NLP solvers are iterative, e.g. SQP $\Rightarrow$ matrices $\nabla c(x,y)$, ... change

- nonlinear effects $\Rightarrow$ replace "LP basis" by KKT system

$$\begin{bmatrix} H_k & -A_k \\ A_k^T & 0 \end{bmatrix}$$

  where $H_k = \nabla^2 \mathcal{L}(x_k, y_k, \lambda_k)$ and $A_k = \nabla c(x_k, y_k)$
- A_k contains active constraint normals
- KKT factors are always out-of-date:
  - factors at $z_k = (x_k, y_k)$ generate step to $z_{k+1} = z_k + d$
  - check convergence at $z_{k+1} \Rightarrow$ overwrite $A_k \leftarrow A_{k+1} \Rightarrow$ factors inconsistent

... not clear how to re-use factors (re-compute?)

Preliminary Numerical Results: NLP vs Hot-QP

CPU times for root node and first set of NLPs

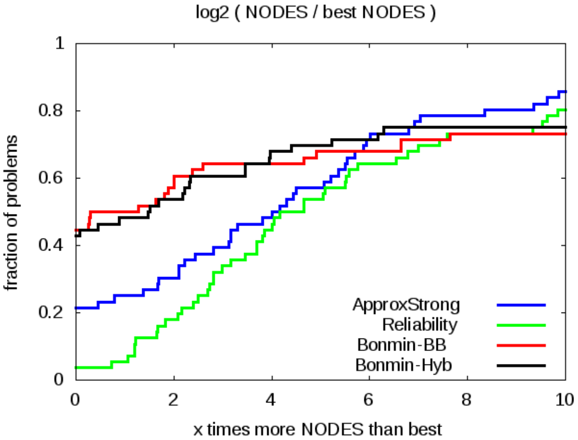
problem	# ints	Full NLP	Single QP	Hot QP
stockcycle	480	4.08	3.32	0.532
RSyn0805H	296	78.7	69.8	1.94
SLay10H	180	18.0	17.8	1.25
Syn30M03H	180	40.9	14.7	2.12

Hot-QP: Solve NLP, re-factor last QP at z_{k+1}, use factors multiple times

- Not clear how good the resulting estimates are
- Loose bounds in tree $\Rightarrow$ re-compute NLP every 10th node?
- Orders of magnitude savings, but many open questions!

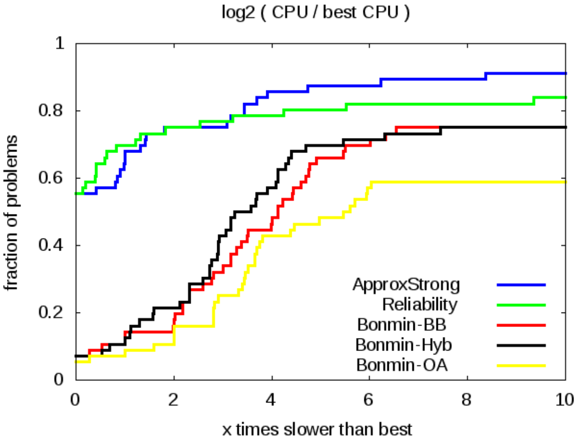
... not yet fully implemented

Comparison to Bonmin (NLP-nodes)



Bonmin solves many fewer NLPs (good if NLPs are hard).

Comparison to Bonmin (CPU-time)



Warm-started NLP pays off hugely (only re-use reduced Hessian)!!!

Conclusions & Future Work

Benefits of Better NLP Solves for MINLP

- reliability & pseudo-cost branching work well
- strong branching too expensive (NLP solves)
- instrument NLP solvers for MINLPs (hot-starts)
- get orders of magnitude speed-up from NLP solvers
- Hot-QP branching: hot-started dual active-set QP

Challenges & Future Work for NLP

- update rhs in hot-started QPs (ideas from SOC steps)
- use inexact NLP solves more in tree-search! ... easy???
- can we use inexact solves to generate cuts? ... hard!!!