Stage

- Quantile regression by random projections
- Forecasting energy prices
- Involves statistics, probability theory, LP
- Implementation: easy
- Taste for theory
- Supported by grants from Siebel Energy Institute and RTE
- Could lead to CIFRE PhD with RTE
- *Hurry if interested*

Mixed-Integer Nonlinear Programming

Leo Liberti, CNRS LIX Ecole Polytechnique
liberti@lix.polytechnique.fr

PMA@MPRO

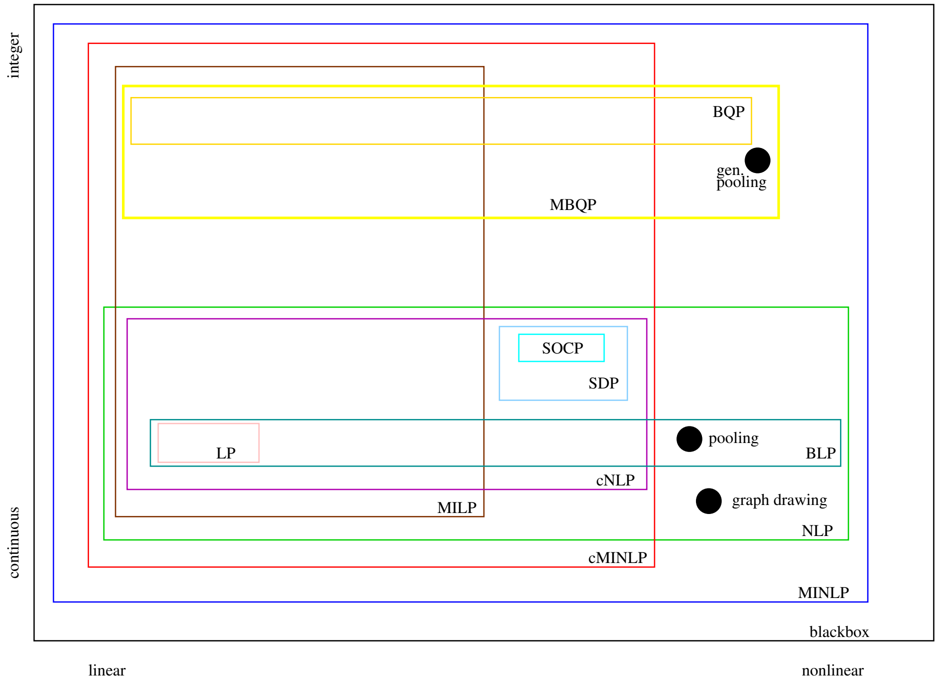
Mathematical Programming Formulations

Mathematical Programming

- MP: formal language for expressing optimization problems P
  - **Parameters** $p =$problem input
    p also called an **instance** of P
  - **Decision variables** x: encode problem output
  - **Objective function** $\min f(p, x)$
  - **Constraints** $\forall i \leq m \quad g_i(p, x) \leq 0$
    f, g: explicit mathematical expressions involving symbols p, x
- If an instance p is given (i.e. an assignment of numbers to the symbols in p is known), write $f(x), g_i(x)$

This excludes black-box optimization

Main optimization problem classes



Notation

- P: MP formulation with decision variables $x = (x_1, \ldots, x_n)$
- **Solution**: assignment of values to decision variables, i.e. a vector $v \in \mathbb{R}^n$
- $\mathcal{F}(P)$ =set of **feasible solutions** $x \in \mathbb{R}^n$ such that $\forall i \leq m \; (g_i(x) \leq 0)$
- $\mathcal{G}(P)$ =set of **globally optimal solutions** $x \in \mathbb{R}^n$ s.t. $x \in \mathcal{F}(P)$ and $\forall y \in \mathcal{F}(P) \; (f(x) \leq f(y))$

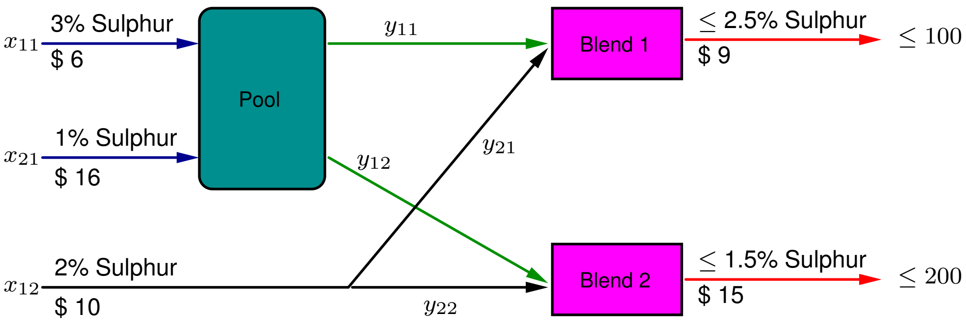
Citations

- Williams, *Model building in mathematical programming*, 2002
- Liberti, Cafieri, Tarissan, *Reformulations in Mathematical Programming: a computational approach*, in Abraham et al. (eds.), Foundations of Comput. Intel., 2009

Haverly's pooling problem

Description

- Given an oil routing network with pools and blenders, unit prices, demands and quality requirements:



- Find the input quantities minimizing the costs and satisfying the constraints: mass balance, sulphur balance, quantity and quality demands

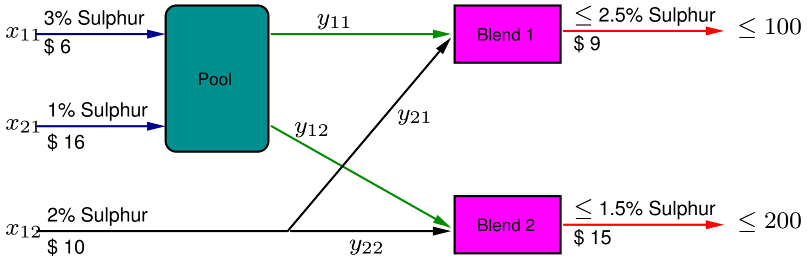
Variables and constraints

- Variables: input quantities x, routed quantities y, percentage p of sulphur in pool
- Every variable must be ≥ 0 (physical quantities)
- Bilinear terms arise to express sulphur quantities in terms of p, y
- Sulphur balance constraint: $3x_{11} + x_{21} = p(y_{11} + y_{12})$
- Quality demands:

$$\begin{aligned} py_{11} + 2y_{21} &\leq 2.5(y_{11} + y_{21}) \\ py_{12} + 2y_{22} &\leq 1.5(y_{12} + y_{22}) \end{aligned}$$

- Continuous bilinear formulation $\Rightarrow$ nonconvex NLP

Formulation



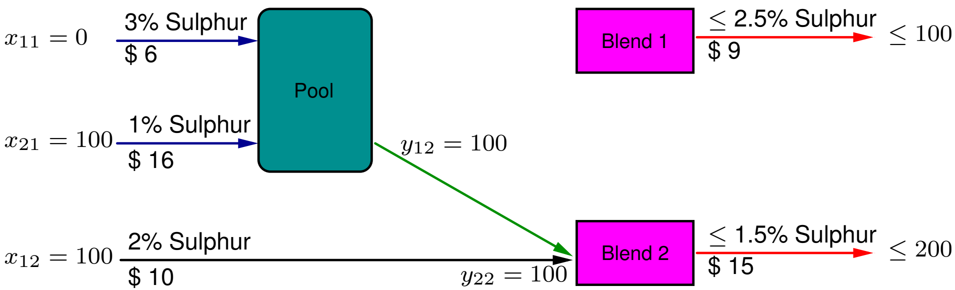
$$
\left\{
\begin{array}{ll}
\min\limits_{x,y,p} & 6x_{11} + 16x_{21} + 10x_{12} - \\
 & \quad -9(y_{11} + y_{21}) - 15(y_{12} + y_{22}) \quad \textit{cost} \\
\text{s.t.} & x_{11} + x_{21} - y_{11} - y_{12} = 0 \quad \textit{mass balance} \\
 & x_{12} - y_{21} - y_{22} = 0 \quad \textit{mass balance} \\
 & y_{11} + y_{21} \leq 100 \quad \textit{demand} \\
 & y_{12} + y_{22} \leq 200 \quad \textit{demand} \\
 & 3x_{11} + x_{21} - p(y_{11} + y_{12}) = 0 \quad \textit{sulphur balance} \\
 & py_{11} + 2y_{21} \leq 2.5(y_{11} + y_{21}) \quad \textit{sulphur limit} \\
 & py_{12} + 2y_{22} \leq 1.5(y_{12} + y_{22}) \quad \textit{sulphur limit}
\end{array}
\right.
$$

Network design

- Decide whether to install pipes or not (0/1 decision)
- Associate a binary variable z_{ij} with each pipe

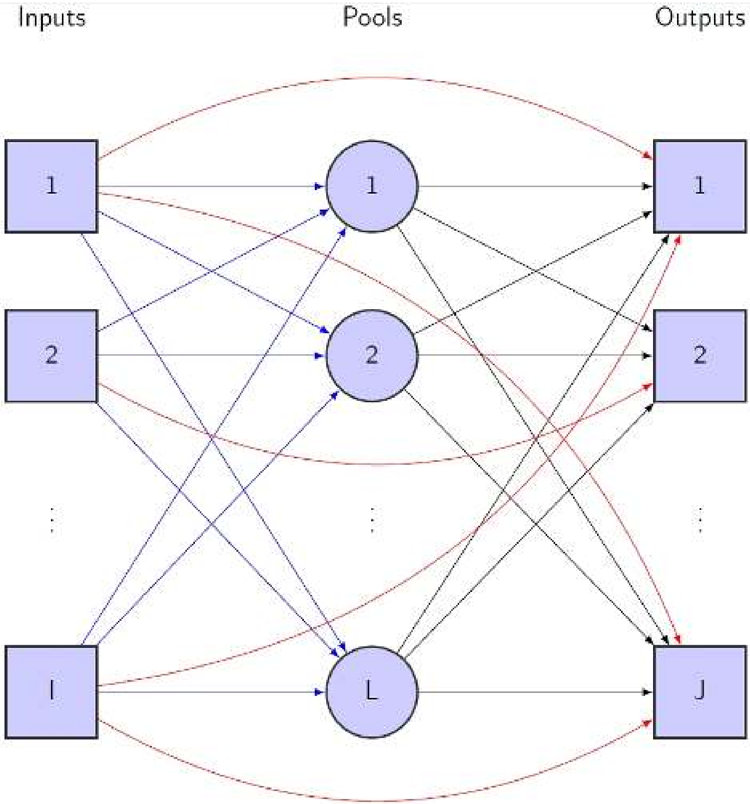
$$
\left\{\begin{array}{rll}
\min\limits_{x,y,p,z} & 6x_{11}+16x_{21}+10x_{12}+\sum_{ij}\theta_{ij}z_{ij}- & \\
 & \quad -9(y_{11}+y_{21})-15(y_{12}+y_{22}) & \textit{cost} \\
\text{s.t.} & x_{11}+x_{21}-y_{11}-y_{12}=0 & \textit{mass balance} \\
 & x_{12}-y_{21}-y_{22}=0 & \textit{mass balance} \\
 & y_{11}+y_{21}\le 100 & \textit{demand} \\
 & y_{12}+y_{22}\le 200 & \textit{demand} \\
\forall i,j\le 2 & y_{ij}\le 200z_{ij} & \textit{pipe activation: if } z_{ij}=0\textit{, no flow} \\
 & 3x_{11}+x_{21}-p(y_{11}+y_{12})=0 & \textit{sulphur balance} \\
 & py_{11}+2y_{21}\le 2.5(y_{11}+y_{21}) & \textit{sulphur limit} \\
 & py_{12}+2y_{22}\le 1.5(y_{12}+y_{22}) & \textit{sulphur limit}
\end{array}\right.
$$

The optimal network



- $z_{11} = 0, \; z_{21} = 0$
- $z_{12} = 1, \; z_{22} = 1$

Pooling problem network



Formulation: sets and parameters

- **Sets**
  - I: index set for input nodes
  - P: index set for pool nodes
  - J: index set for output nodes
  - K: index set for flow attributes
  - $\forall p \in P\ N^{-}(p)$: index set for inputs $\to p$
  - $\forall p \in P\ N^{+}(p)$: index set for $p \to$ outputs
- **Parameters**
  - $\forall i \in I\ S_i =$ supply at node i
  - $\forall j \in J\ D_i =$ max. demand at node j
  - $\forall i \in I, k \in K\ A_{ik} =$ qty of attribute k in input flow i
  - $\forall j \in J, k \in K\ L_{jk} =$ min qty attr k at output j
  - $\forall j \in J, k \in K\ U_{jk} =$ max qty attr k at output j
  - $\forall i \in I\ c_i^I =$ unit acquisition costs at input i
  - $\forall j \in J\ c_j^J =$ unit selling price at output j

Formulation: decision variables & objective

- **Decision variables**
  - $\forall i \in I, p \in P \; x_{ip} =$ **flow in pipe** (i,p)
  - $\forall p \in P, j \in J \; y_{ip} =$ **flow in pipe** (p,j)
  - $\forall p \in P, k \in K \; q_{pk} =$ **qty attr** k **in pool** p
  - $\forall j \in J, k \in K \; Q_{jk} =$ **qty attr** k **in output** j
- **Objective function**

$$\min F(x,y) = \sum_{\substack{p \in P \\ i \in N^-(p)}} c_i^I x_{ip} - \sum_{\substack{p \in P \\ j \in N^+(p)}} c_j^J y_{pj}$$

Formulation: constraints

- *Supply:* $\forall i \in I \quad \sum_{\substack{p \in P \\ i \in N^-(p)}} x_{ip} \leq S_i$
- *Max demand:* $\forall j \in J \quad \sum_{\substack{p \in P \\ j \in N^+(p)}} y_{pj} \leq D_j$
- *Mass balance for flow across pools:*

$$\forall p \in P \qquad \sum_{i \in N^-(p)} x_{ip} = \sum_{j \in N^+(p)} y_{pj}$$

- *Attr. qty balance input/pools:*

$$\forall p \in P, k \in K \qquad \sum_{i \in N^-(p)} A_{ik} x_{ip} = \sum_{i \in N^-(p)} q_{pk} x_{ip}$$

- *Attr. qty balance pools/output:*

$$\forall j \in J, k \in K \qquad \sum_{\substack{p \in P \\ j \in N^+(p)}} q_{pk} y_{pj} = \sum_{\substack{p \in P \\ j \in N^+(p)}} Q_{jk} y_{pj}$$

Generalized pooling problem

- **Decision variables**
  - $\forall i \in I, p \in P \; z^{\mathsf{in}}_{ip} = 1$ **iff pipe** (i,p) **installed,** 0 **othw**
  - $\forall p \in P, j \in J \; z^{\mathsf{out}}_{pj} = 1$ **iff pipe** (p,j) **installed,** 0 **othw**
- **Objective function**

$$\min F(x,y) + \sum_{\substack{p \in P \\ i \in N^-(p)}} z^{\mathsf{in}}_{ip} + \sum_{\substack{p \in P \\ j \in N^+(p)}} z^{\mathsf{out}}_{pj}$$

- **Constraints**
  - *Pipe activation:*

$$\begin{array}{lrcl} \forall p \in P, i \in N^-(p) & x_{ip} & \leq & S_i z^{\mathsf{in}}_{ip} \\ \forall p \in P, j \in N^+(p) & y_{pj} & \leq & D_j z^{\mathsf{out}}_{pj} \end{array}$$

Classification in systematics

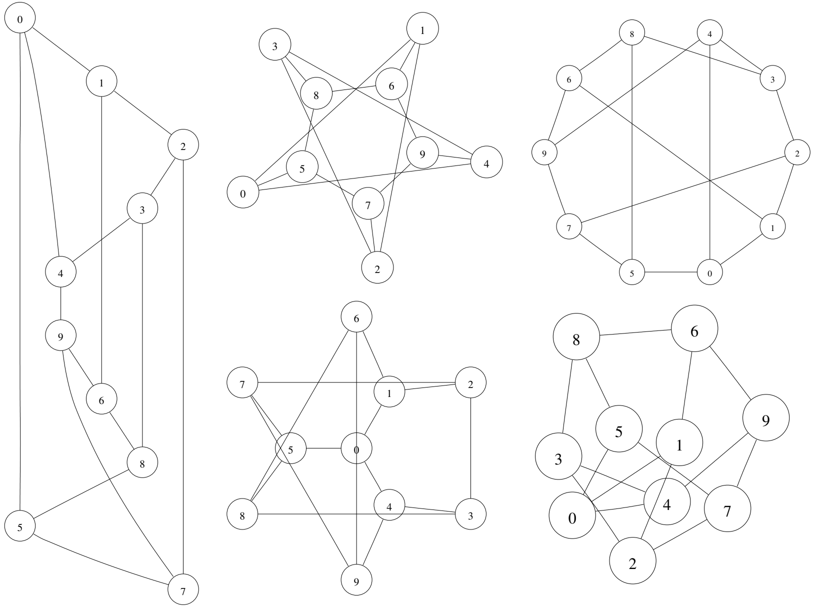
- Attribute constraints involve $q_{pk}x_{ip}, q_{pk}y_{pj}, Q_{jk}y_{pj}$
- Bilinear terms in equations: **nonconvex** $\mathcal{F}(P)$
- $\Rightarrow$ **(nonconvex) NLP**
- *Generalized pooling problem*: **(nonconvex) MINLP**

Citations

1. C. Haverly, *Studies of the behaviour of recursion for the pooling problem*, ACM SIGMAP Bulletin, 1978
2. Adhya, Tawarmalani, Sahinidis, *A Lagrangian approach to the pooling problem*, Ind. Eng. Chem., 1999
3. Audet et al., *Pooling Problem: Alternate Formulations and Solution Methods*, Manag. Sci., 2004
4. Liberti, Pantelides, *An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms*, JOGO, 2006
5. Misener, Floudas, *Advances for the pooling problem: modeling, global optimization, and computational studies*, Appl. Comput. Math., 2009
6. D'Ambrosio, Linderoth, Luedtke, *Valid inequalities for the pooling problem with binary variables*, LNCS, 2011

Drawing graphs

At a glance



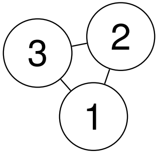
Which graph has most symmetries?

Euclidean graphs

- Graph $G = (V, E)$, edge weight function $d : E \to \mathbb{R}_+$
- E.g. $V = \{1, 2, 3\}$, $E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$
  $d_{12} = d_{13} = d_{23} = 1$
- Find positions $x_v = (x_{v1}, x_{v2})$ of each $v \in V$ in the plane s.t.:

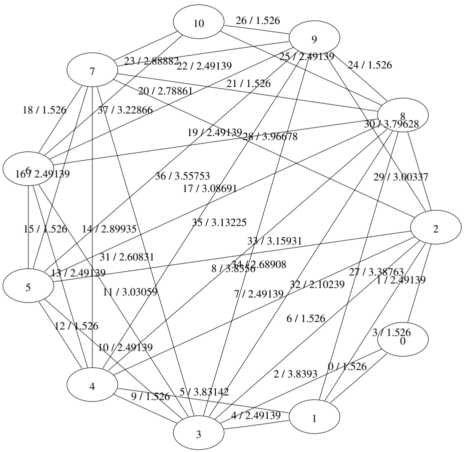
$$\forall \{u, v\} \in E \quad \|x_u - x_v\|_2 = d_{uv}$$

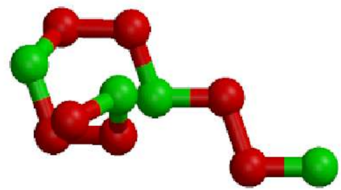
- Generalization to $\mathbb{R}^K$ for $K \in \mathbb{N}$: $x_v = (x_{v1}, \ldots, x_{vK})$



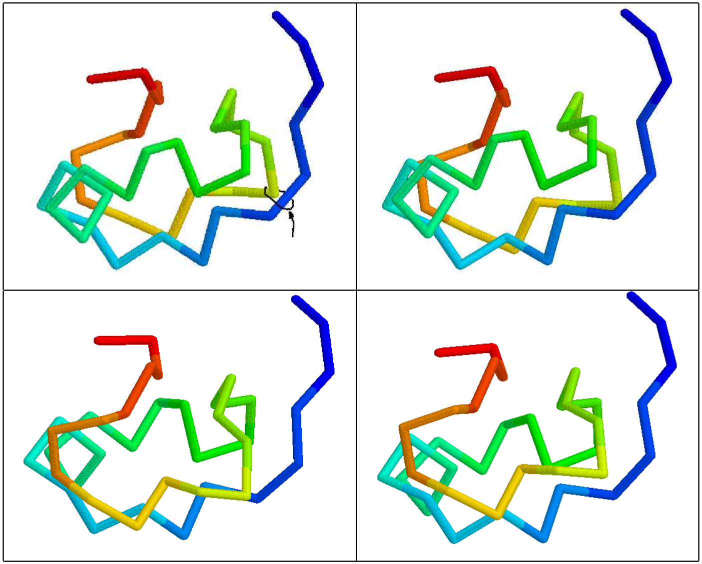
Application to proteomics

An artificial protein test set: `lavor-11_7`



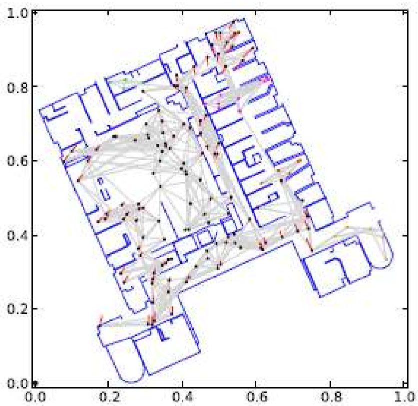


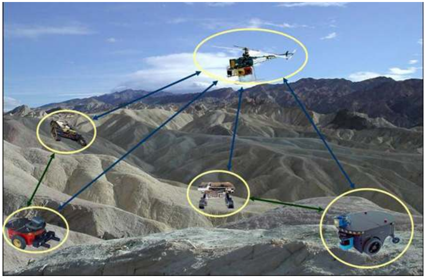
Embedding protein data in $\mathbb{R}^3$



`1aqr`: **four non-isometric embeddings**

Sensor networks in 2D and 3D





Formulation

$$\min_{x,t} \sum_{\{u,v\}\in E} t_{uv}^2$$

$$\forall \{u,v\} \in E \quad \sum_{k\le K} (x_{uk} - x_{vk})^2 \;=\; d_{uv}^2 + t_{uv}$$

Citations

1. Lavor, Liberti, Maculan, Mucherino, *Recent advances on the discretizable molecular distance geometry problem*, Eur. J. of Op. Res., invited survey
2. Liberti, Lavor, Mucherino, Maculan, *Molecular distance geometry methods: from continuous to discrete*, Int. Trans. in Op. Res., **18**:33-51, 2010
3. Liberti, Lavor, Maculan, *Computational experience with the molecular distance geometry problem*, in J. Pintér (ed.), *Global Optimization: Scientific and Engineering Case Studies*, Springer, Berlin, 2006

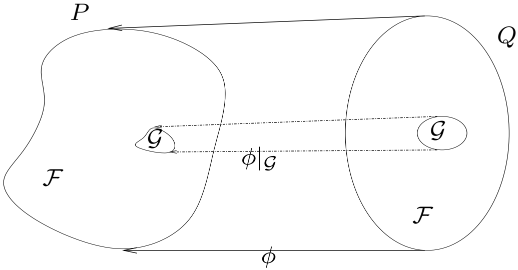
Reformulations

Exact reformulations

- The formulation Q is an **exact reformulation** of P if

  $\exists$ an efficiently computable surjective map $\phi : \mathcal{F}(Q) \to \mathcal{F}(P)$ s.t. $\phi|_{\mathcal{G}(Q)}$ is onto $\mathcal{G}(P)$

- Informally: *any optimum of Q can be mapped easily to an optimum of P, and for any optimum of P there is a corresponding optimum of Q*



- Construct Q so that it is easier to solve than P

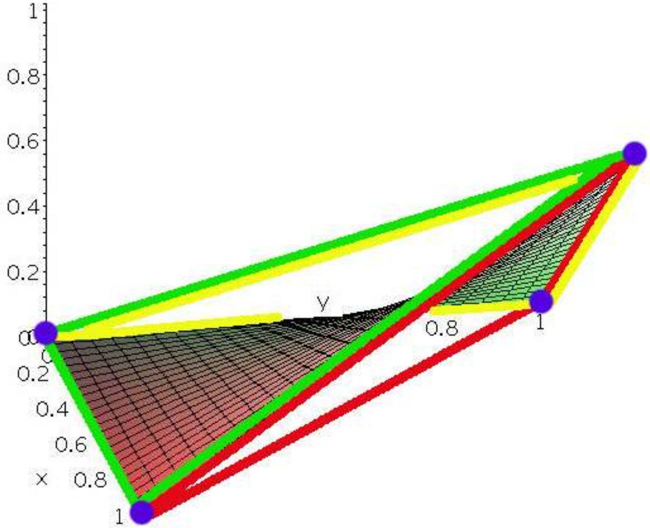
xy when x is binary

- If $\exists$ bilinear term xy where $x \in \{0, 1\}$, $y \in [0, 1]$
- We can construct an **exact reformulation**:
  - Replace each term xy by an added variable w
  - Adjoin Fortet's reformulation constraints:

$$
\begin{aligned}
w &\geq 0 \\
w &\geq x + y - 1 \\
w &\leq x \\
w &\leq y
\end{aligned}
$$

  - Get a MILP reformulation
  - Solve reformulation using CPLEX: more effective than solving MINLP

“Proof”



Relaxations

- The formulation Q is a **relaxation** of P if $\min f_Q(y) \leq \min f_P(x) \quad (*)$
- Relaxations are used to compute **worst-case bounds** to the optimum value of the original formulation
- Construct Q so that it is easy to solve
- Proving $(*)$ may not be easy in general
- **The usual strategy**:
  - Make sure $y \supset x$ and $\mathcal{F}(Q) \supseteq \mathcal{F}(P)$
  - Make sure $\forall x \in \mathcal{F}(P)\ (f_Q(y) \leq f_P(x))$
  - Then it follows that Q is a relaxation of P
- Example: *convex relaxation*
  - $\mathcal{F}(Q)$ a convex set containing $\mathcal{F}(P)$
  - f_Q a convex underestimator of f_P
  - Then Q is a cNLP and can be solve efficiently

xy when x, y continuous

- Get bilinear term xy where $x \in [x^L, x^U]$, $y \in [y^L, y^U]$
- We can construct a **relaxation**:
  - Replace each term xy by an added variable w
  - Adjoin following constraints:

$$
\begin{aligned}
w &\geq x^L y + y^L x - x^L y^L \\
w &\geq x^U y + y^U x - x^U y^U \\
w &\leq x^U y + y^L x - x^U y^L \\
w &\leq x^L y + y^U x - x^L y^U
\end{aligned}
$$

- These are called **McCormick's envelopes**
- Get an LP relaxation (solvable in polynomial time)

Software

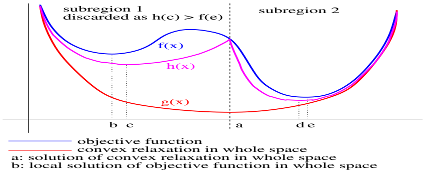
- ROSE (`https://projects.coin-or.org/ROSE`)

Citations

- McCormick, *Computability of global solutions to factorable nonconvex programs: Part I — Convex underestimating problems*, Math. Prog. 1976
- Liberti, *Reformulations in Mathematical Programming: definitions and systematics*, RAIRO-RO 2009

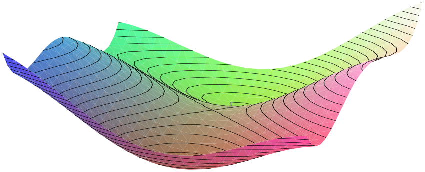
Global Optimization methods

Deterministic / Stochastic



Exact = Deterministic

- "Exact" in continuous space: ε-approximate (*find solution within pre-determined* ε *distance from optimum in obj. fun. value*)
- For some problems, finite convergence to optimum ($\varepsilon = 0$)



Heuristic = Stochastic

- Find solution with probability 1 in infinite time

Multistart

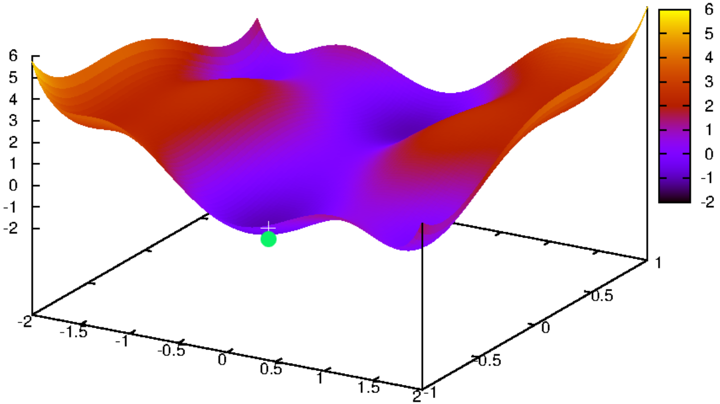
- The easiest GO method

$f^* = \infty$
$x^* = (\infty, \ldots, \infty)$
while $\neg$ termination **do**
    $x' = (\text{random}(), \ldots, \text{random}())$
    $x = \text{localSolve}(P, x')$
    if $f_P(x) < f^*$ **then**
        $f^* \leftarrow f_P(x)$
        $x^* \leftarrow x$
    end if
end while

- Termination condition: e.g. *repeat* k *times*

Six-hump camelback function

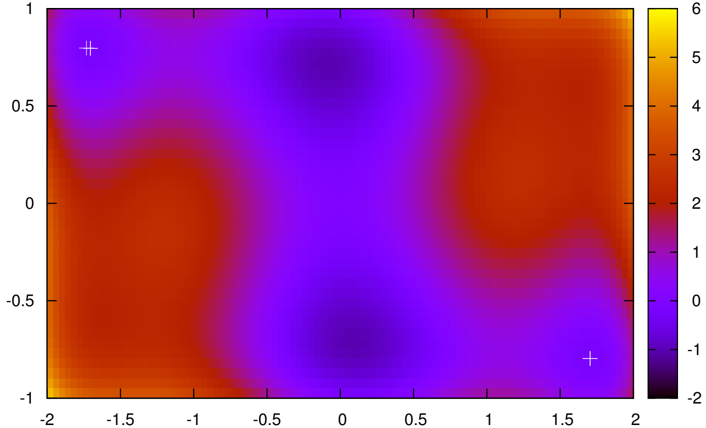
$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Global optimum (COUENNE)

Six-hump camelback function

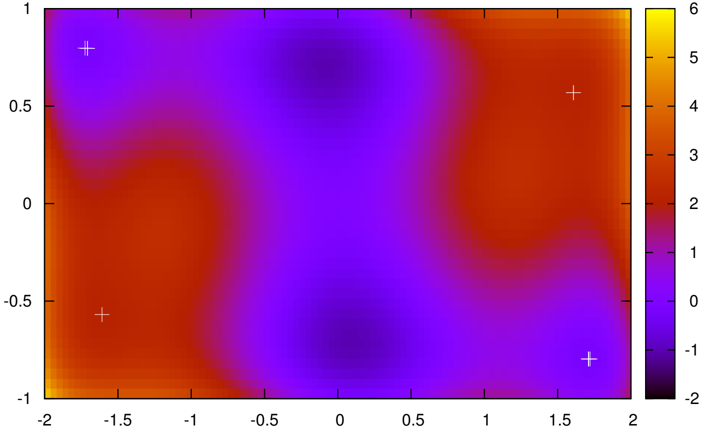
$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Multistart with IPOPT, $k = 5$

Six-hump camelback function

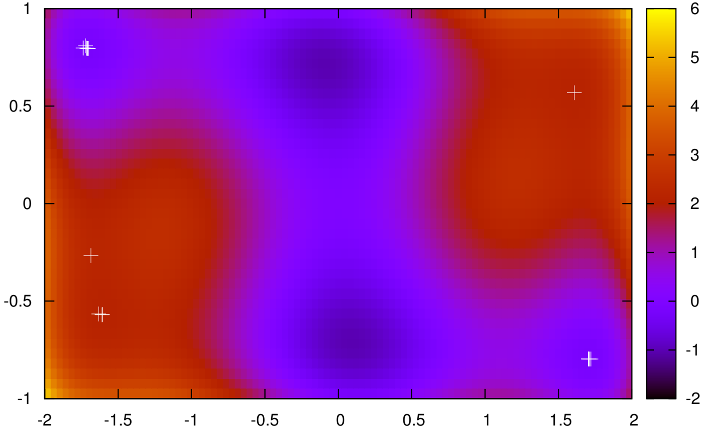
$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Multistart with IPOPT, $k = 10$

Six-hump camelback function

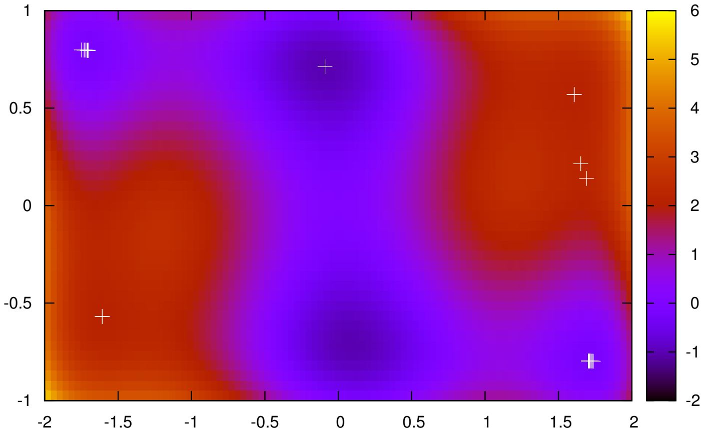
$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Multistart with IPOPT, $k = 20$

Six-hump camelback function

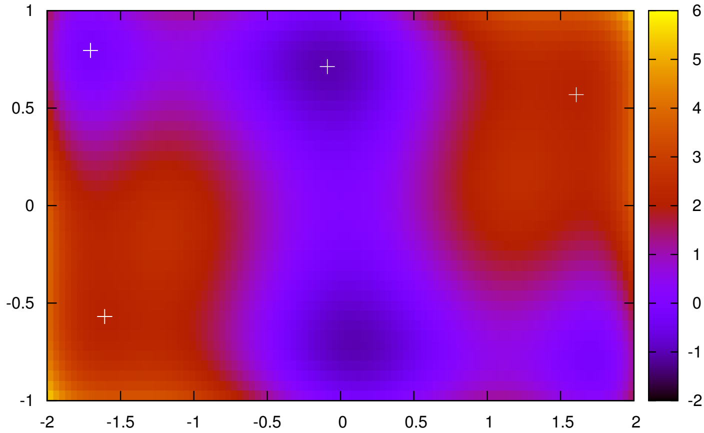
$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Multistart with IPOPT, $k = 50$

Six-hump camelback function

$$f(x,y) = 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$



Multistart with SNOPT, $k = 20$

Citations

- Schoen, *Two-Phase Methods for Global Optimization*, in Pardalos et al. (eds.), *Handbook of Global Optimization 2*, 2002
- Liberti, Kucherenko, *Comparison of deterministic and stochastic approaches to global optimization*, ITOR 2005

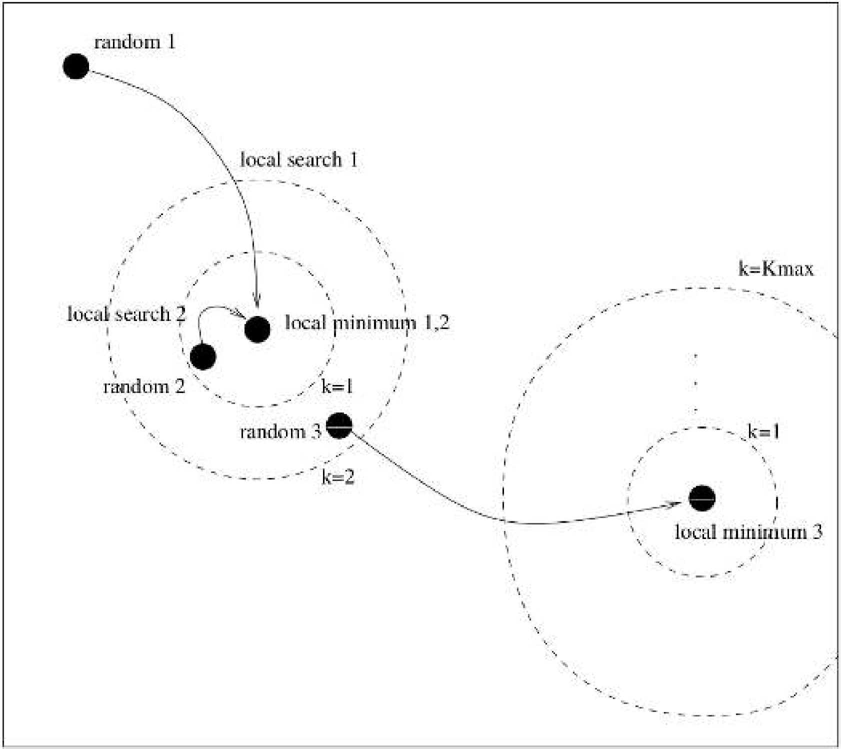
Section 2

Variable Neighbourhood Search

Variable Neighbourhood Search

- Applicable to discrete and continuous problems
- Uses any local search as a black-box
- In its basic form, easy to implement
- Few configurable parameters
- Structure of the problem dealt with by local search
- Few lines of code around LS black-box

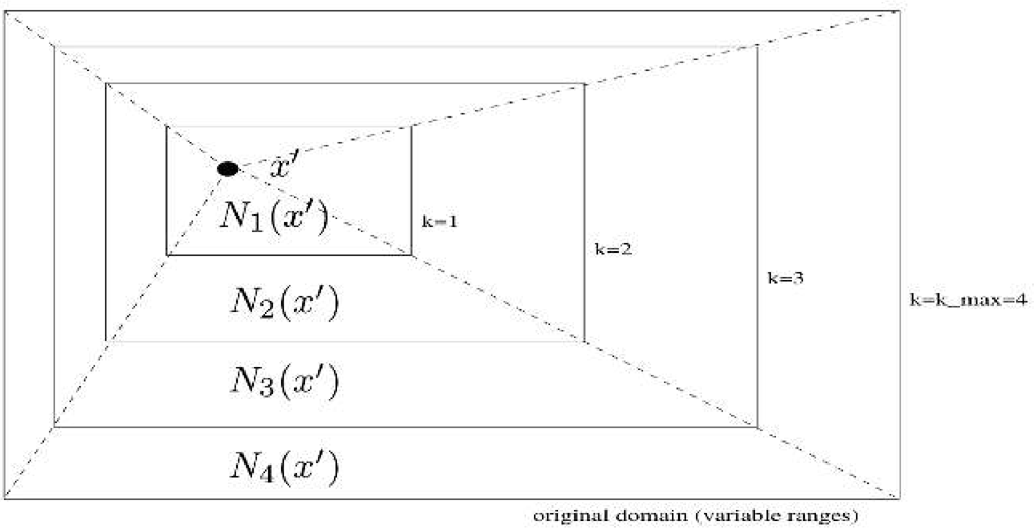
Variable Neighbourhood Search



Variable Neighbourhood Search

Input: max no. $k_{\max}$ of neighbourhoods
loop
    $k \leftarrow 1$, sample rnd. pt. $\tilde{x}$, LocSearch$(\tilde{x}) = x^*$
    while $k \leq k_{\max}$ **do**
        $N_k(x^*)$ neighb. of x^* s.t. $N_k(x^*) \supset N_{k-1}(x^*)$
        sample rnd. pt. $\tilde{x}$ from $N_k(x^*)$
        LocSearch$(\tilde{x}) = x'$
        if x' better than x^* **then**
            $x^* \leftarrow x'$, $k \leftarrow 0$
        end if
        $k \leftarrow k + 1$
        if termination condition, then exit
    end while
end loop

Neighbourhood structure (continuous vars)



Neighbourhood structure (binary vars)

- $y \in \{0,1\}^p$ **binary vars**
- **current incumbent** $y^* \in \{0,1\}^p$
- **"neighbourhood" centered at** y^* **of radius** $k \in \mathbb{N}$**:**

$$\sum_{\substack{i \le p \\ y^*_i = 0}} y_i + \sum_{\substack{i \le p \\ y^*_i = 1}} (1 - y_i) \le k$$

- *Local Branching constraint*
  allows at most k **flips on** p **bin vars**

Citations

1. **L. Liberti**, M. Dražić, *Variable Neighbourhood Search for the Global Optimization of Constrained NLPs*, Proc. of the Global Optimization Workshop, Almeria, Spain, 18-22 September 2005
2. **L. Liberti**, N. Mladenović, G. Nannicini, *A recipe for finding good solutions to MINLPs*, Mathematical Programming Computation, **3**:349-390, 2011

spatial Branch-and-Bound (sBB)

Generalities

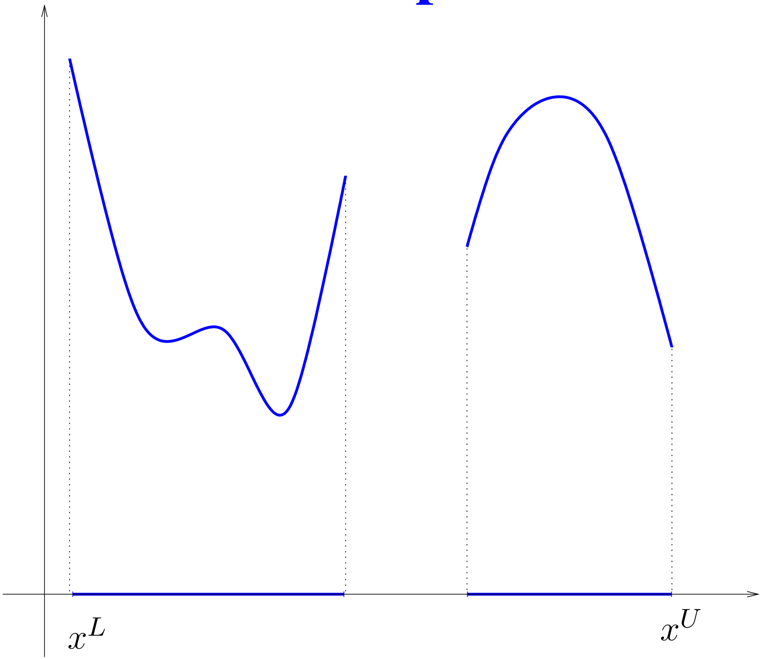
- Tree-like search
- Explores search space exhaustively but implicitly
- Builds a sequence of decreasing upper bounds and increasing lower bounds to the global optimum
- Exponential worst-case
- Only general-purpose “exact” algorithm for MINLP

  Since continuous vars are involved, should say “ε-approximate”

- Like BB for MILP, but may branch on continuous vars

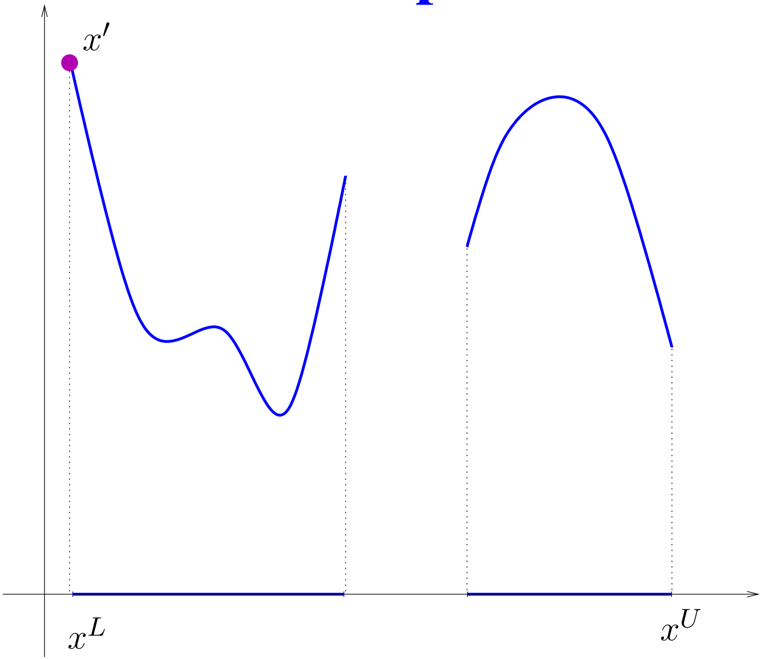
  Done whenever one is involved in a nonconvex term

Example



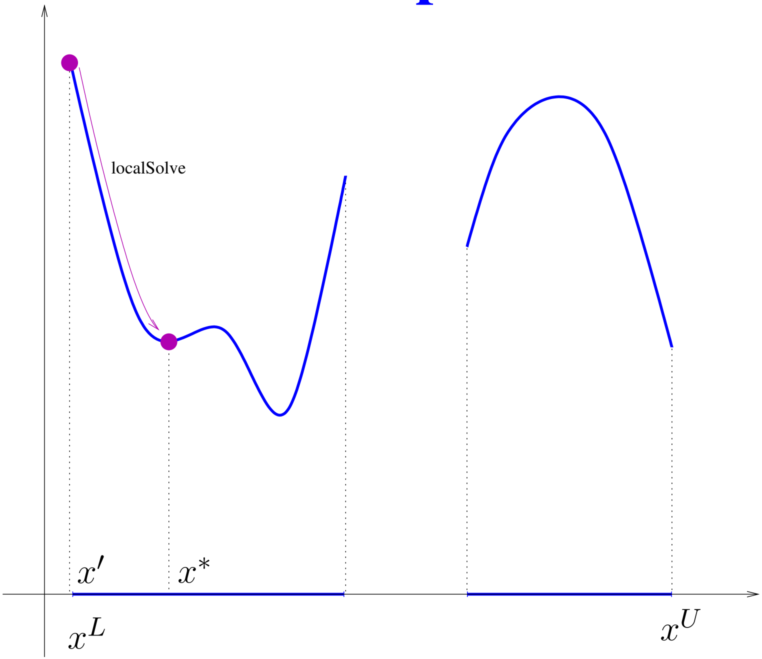
Original problem P

Example



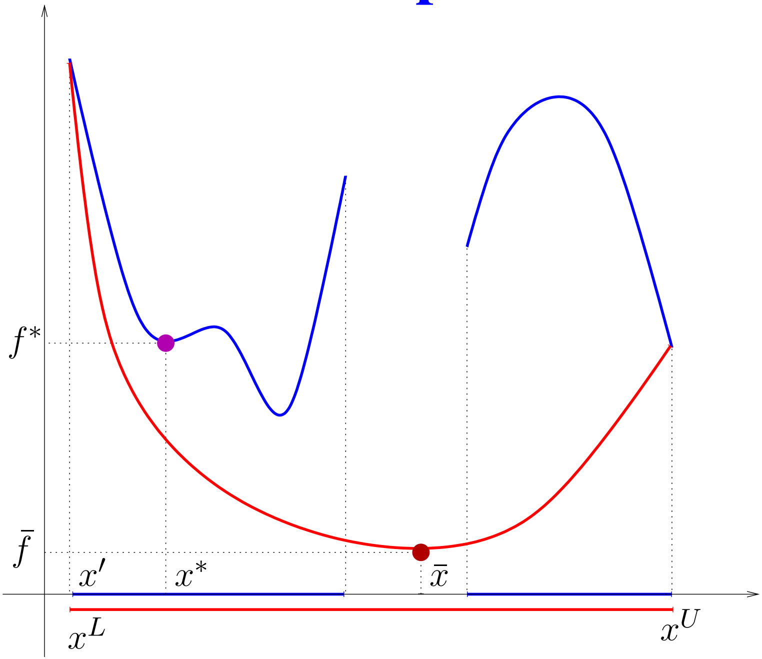
Starting point x'

Example



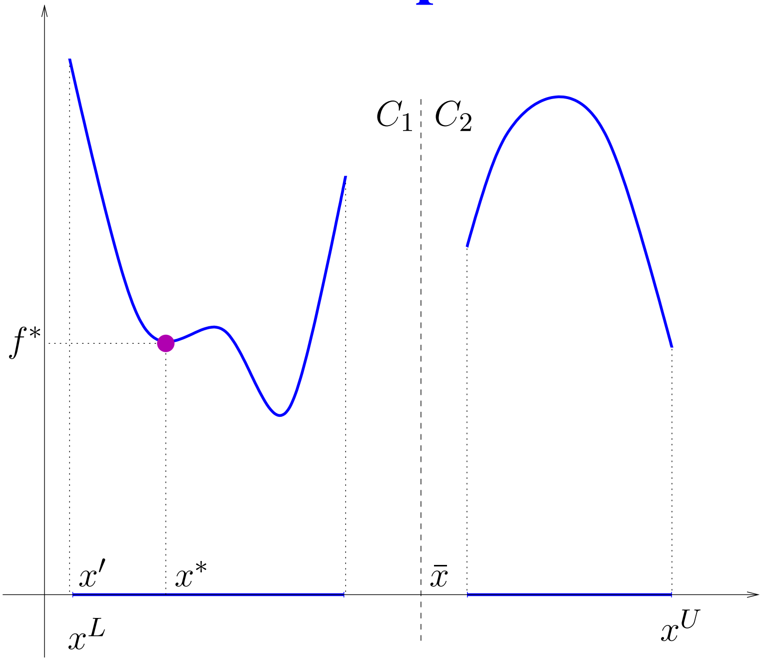
Local (upper bounding) solution x^*

Example



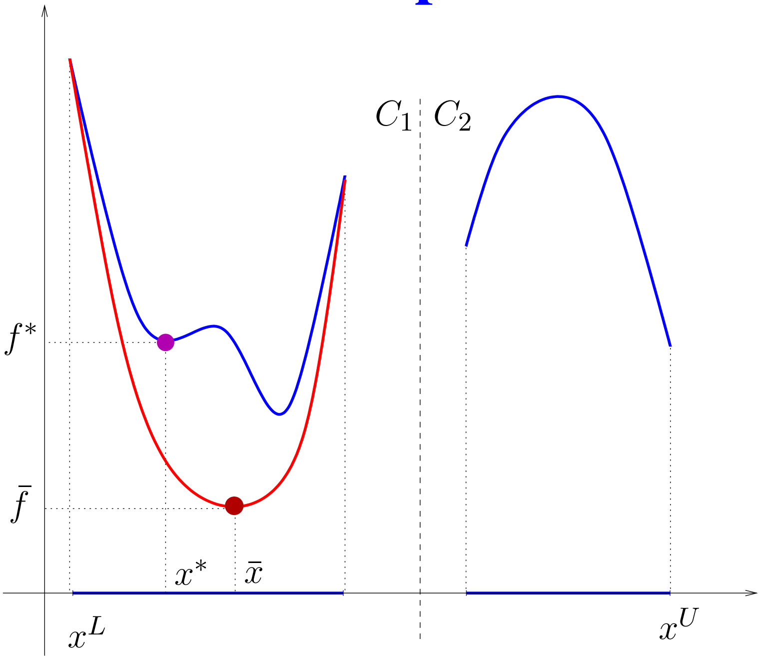
Convex relaxation (lower) bound $\bar{f}$ *with* $|f^* - \bar{f}| > \varepsilon$

Example



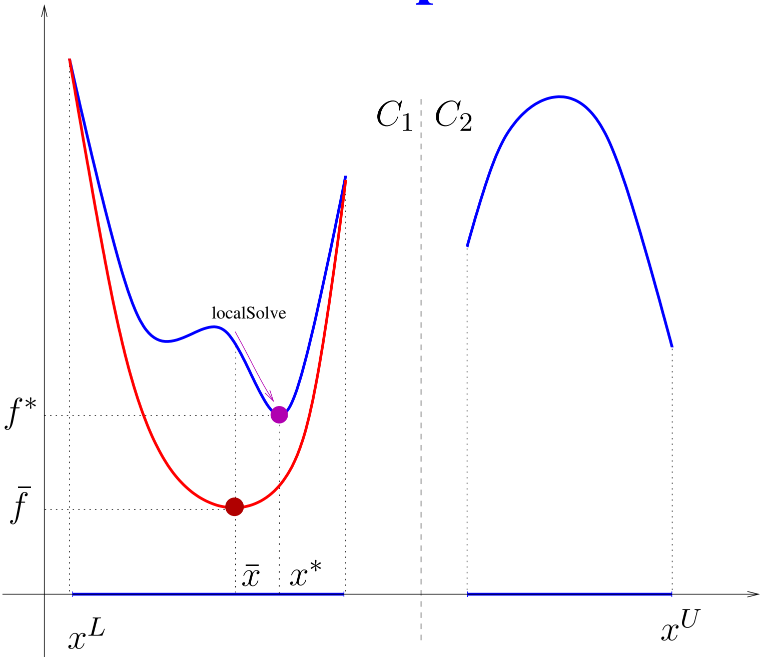
Branch at $x = \bar{x}$ *into* C_1, C_2

Example



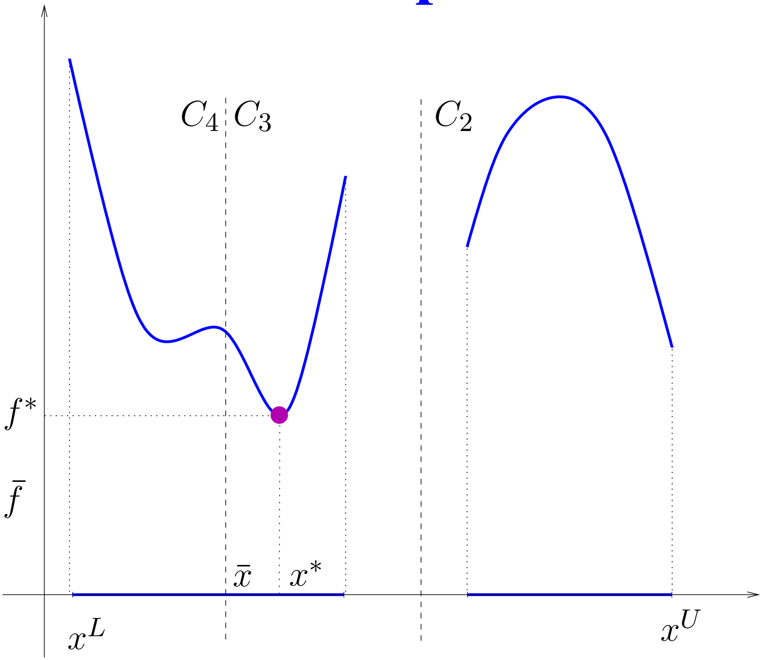
Convex relaxation on C_1: lower bounding solution $\bar{x}$

Example



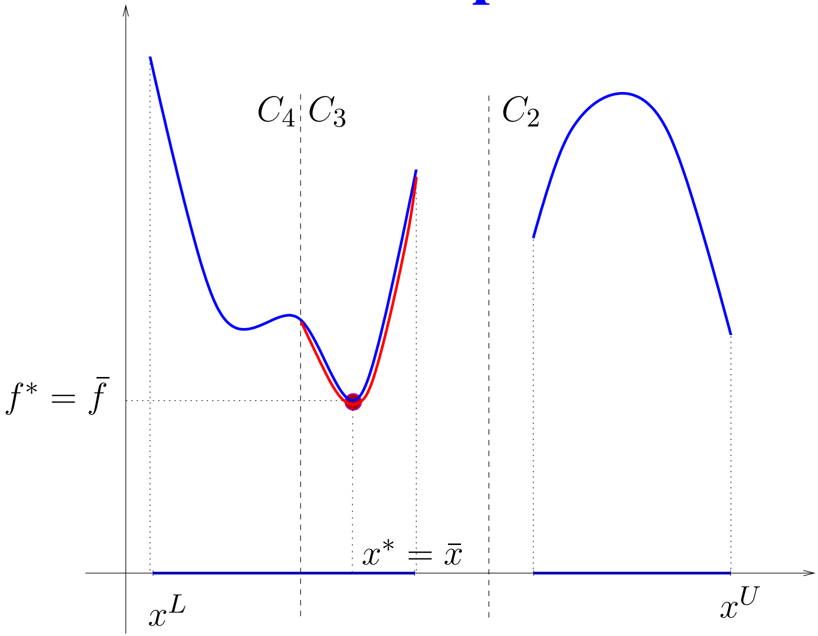
localSolve. *from $\bar{x}$: new upper bounding solution x^**

Example



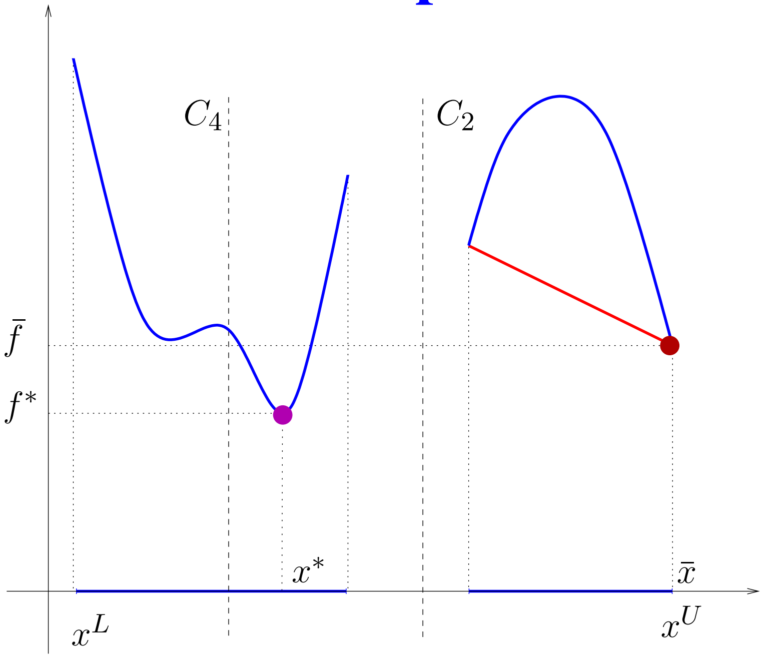
$|f^* - \bar{f}| > \varepsilon$: *branch at* $x = \bar{x}$

Example



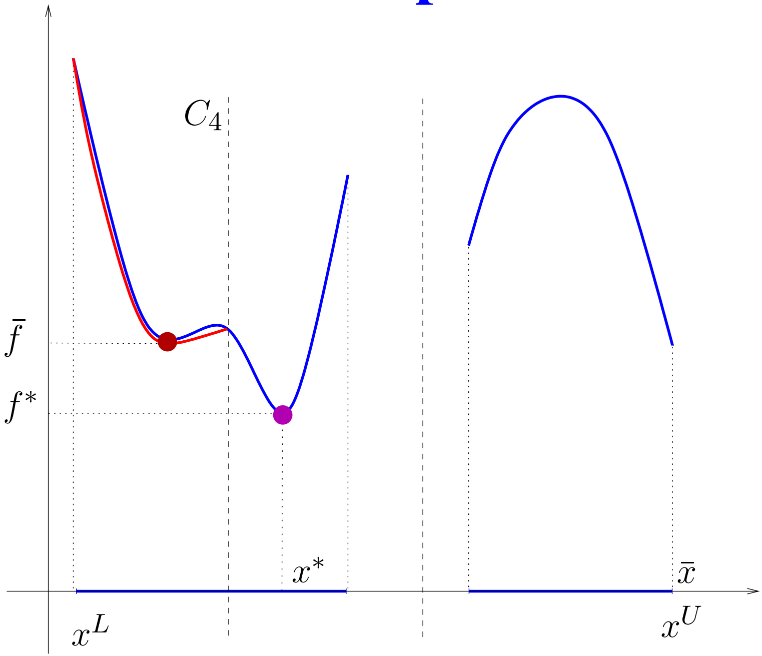
Repeat on C_3: get $\bar{x} = x^*$ and $|f^* - \bar{f}| < \varepsilon$, no more branching

Example



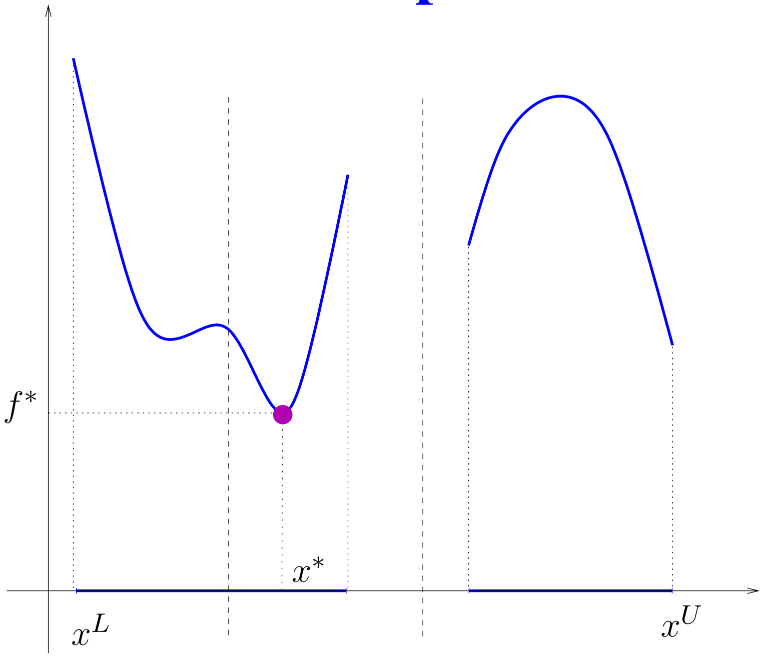
Repeat on C_2: $\bar{f} > f^*$ (can't improve x^* in C_2)

Example



Repeat on C_4*:* $\bar{f} > f^*$ *(can't improve* x^* *in* C_4*)*

Example



No more subproblems left, return x^* and terminate

Pruning

1. P was branched into C_1, C_2
2. C_1 was branched into C_3, C_4
3. C_3 was **pruned by optimality**
   ($x^* \in \mathcal{G}(C_3)$ was found)
4. C_2, C_4 were **pruned by bound**
   (lower bound for C_2 worse than f^*)
5. No more nodes: whole space explored, $x^* \in \mathcal{G}(P)$

- Search generates a tree
- Suproblems are nodes
- Nodes can be pruned by optimality, bound or **infeasibility** (when subproblem is infeasible)
- Otherwise, they are branched

Logical flow

Notation:

- $C = P[x^L, x^U]$ is P restricted to $x \in [x^L, x^U]$
- x^*: best optimum so far (start with $x^* = \infty$)

- C **could be feasible or infeasible**
  - If C is feasible, **we might find a glob. opt.** x' **of** C **or not**
    - If we find **glob. opt.** x' **improving** x^*, update $x^* \leftarrow x'$
    - Else, **try and show no point in** $\mathcal{F}(C)$ **improves** x^*
      - Else **branch** C into two suproblems and recurse on each

        subproblems have smaller feasible regions $\Rightarrow$ *"easier"*
  - Else C **is infeasible, discard**

Correctness

- Look at else cases:
  - C infeasible $\Rightarrow$ can discard C
  - C feasible and no point $\mathcal{F}(C)$ improves x^* $\Rightarrow$ can discard C
- Branching $\Rightarrow$ any subproblem that we're NOT sure could improve x^* is considered again later
- $\Rightarrow$ If process terminates, we'll have explored all those parts of $\mathcal{F}(P)$ that can contain an optimum better than x^*
  - If $x^* = \infty$, P infeasible, otherwise $x^* \in \mathcal{G}(P)$
  - Might fail to terminate if $\varepsilon = 0$

A recursive version

processSubProblem$_\varepsilon(C)$:

if isFeasible(C) **then**
  $x' =$ globalOpt(C)
  if $x' \neq \infty$ **then**
    if $f_P(x') < f_P(x^*)$ **then**
      update $x^* \leftarrow x'$ `// improvement`
    end if
  else
    if lowerBound$(C) < f_P(x^*) - \varepsilon$ **then**
      Split $[x^L, x^U]$ into two hyperrectangles $[x^L, \tilde{x}]$, $[\underline{x}, x^U]$
      processSubProblem$_\varepsilon(C[x^L, \tilde{x}])$
      processSubProblem$_\varepsilon(C[\underline{x}, x^U])$
    end if
  end if
end if

Bad news

1. If globalOpt(C) works on any problem, why not call globalOpt(P) and be done with it?
2. For arbitrary C, isFeasible(C) is **undecidable**
3. How do we compute lowerBound(C)?

Upper bounds

Upper bounds: x^* can only decrease

- Computing the global optima for each subproblem yields candidates for updating x^*
- As long as we only update x^* when x' improves it, we don't need x' to be a *global* optimum
- Any "good feasible point" will do
- Specifically, use *feasible local optima*
- $\Rightarrow$ Replace $\text{globalOpt}()$ by $\text{localSolve}()$

Lower bound

Lower bounds: increase over $\supset$-chains

- Let R_P be a relaxation of P such that:
  1. R_P also involves the decision variables of P (*and perhaps some others*)
  2. for any range $I = [x^L, x^U]$,

     $R_P[I]$ is a relaxation of $P[I]$
  3. if I, I' are two ranges

     $I \supseteq I' \to \min R_P[I] \leq \min R_P[I']$
  4. For any subproblem C of P,

     finding $x \in \mathcal{G}(R_C)$ or showing $\mathcal{F}(R_C) = \varnothing$ is efficient

     Specifically, $\bar{x} = \text{localSolve}(R_C) \in \mathcal{G}(R_C)$
- Define $\text{lowerBound}(C) = f_{R_C}(\bar{x})$

A decidable feasibility test

- Processing C when it's infeasible will make sBB slower but not incorrect
- $\Rightarrow$ sBB still works if we simply **never discard a potentially feasible** C
- Use a "partial feasibility test" isEvidentlyInfeasible(P)
  - If isEvidentlyInfeasible(C) is `true`, then C is **guaranteed** to be infeasible, and we can discard it
  - Otherwise, we simply don't know, and we shall process it
- **Thm**: If R_C is infeasible then C is infeasible
- **Proof**: $\varnothing = \mathcal{F}(R_C) \supseteq \mathcal{F}(C) = \varnothing$
- isEvidentlyInfeasible$(C) = \begin{cases} \texttt{true} & \text{if localSolve}(R_C) = \infty \\ \texttt{false} & \text{otherwise} \end{cases}$

Choice of best next node

- Instead recursion order, process first nodes which are more likely to yield a glob. opt.
- **Advantages**
  - Glob. opt. of P found early
    $\Rightarrow$ *easier to prune by bound*
  - If sBB stopped early, more chance that $x^* \in \mathcal{G}(P)$
- Indication of a "good subproblem": **if lower bound is lowest**
- Store subproblems in a min-priority queue $\mathcal{Q}$, where priority(C) is given by a lower bound for C

Software

- COUENNE (open source, AMPL interface) (`projects.coin-or.org/Couenne`)
- GlobSol (open source, interval arithmetic bounds) (`http://interval.louisiana.edu/GLOBSOL/`)
- BARON (commercial, GAMS interface)
- LGO (commercial, Lipschitz constant bounds)
- LINDOGLOBAL (commercial)
- Some research codes (αBB, *oo*OPS, LaGO, GLOP, Coconut)

Citations

- Falk, Soland, *An algorithm for separable nonconvex programming problems*, Manag. Sci. 1969
- Horst, Tuy, *Global Optimization*, Springer 1990
- Adjiman, Floudas et al., *A global optimization method, αBB, for general twice-differentiable nonconvex NLPs*, Comp. Chem. Eng. 1998
- Ryoo, Sahinidis, *Global optimization of nonconvex NLPs and MINLPs with applications in process design*, Comp. Chem. Eng. 1995
- Smith, Pantelides, *A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs*, Comp. Chem. Eng. 1999
- Nowak, *Relaxation and decomposition methods for Mixed Integer Nonlinear Programming*, Birkhäuser, 2005
- Belotti, Liberti et al., *Branching and bounds tightening techniques for nonconvex MINLP*, Opt. Meth. Softw., 2009

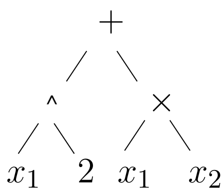
To make an sBB work efficiently, you need further tricks

Expression trees

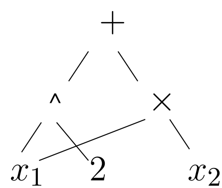
Representation of objective f and constraints g

Encode mathematical expressions in trees or DAGs

E.g. $x_1^2 + x_1 x_2$:



tree



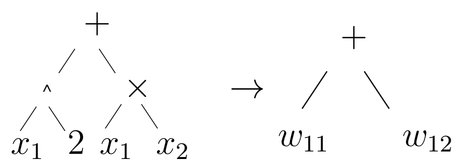
DAG

Standard form

- Identify all nonlinear terms $x_i \otimes x_j$, replace them with a linearizing variable w_{ij}
- Add a *defining constraint* $w_{ij} = x_i \otimes x_j$ to the formulation
- Standard form:

$$\left.\begin{array}{rrcl} \min & c^\top(x,w) & & \\ \text{s.t.} & A(x,w) & \begin{smallmatrix}\leq\\=\\\geq\end{smallmatrix} & b \\ & w_{ij} & = & x_i \otimes_{ij} x_j \text{ for suitable } i,j \\ & \text{bounds} & \& & \text{integrality constraints} \end{array}\right\}$$

- $x_1^2 + x_1 x_2 \Rightarrow \left\{\begin{array}{c} w_{11} + w_{12} \\ w_{11} = x_1^2 \\ w_{12} = x_1 x_2 \end{array}\right.$:

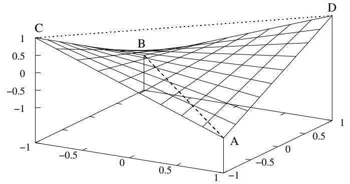


Convex relaxation

- *Standard form*: all nonlinearities in defining constraints
- Each defining constraint $w_{ij} = x_i \otimes x_j$ is replaced by two convex inequalities:

$$\begin{aligned} w_{ij} &\le \text{overestimator}(x_i \otimes x_j) \\ w_{ij} &\ge \text{underestimator}(x_i \otimes x_j) \end{aligned}$$

- E.g. convex/concave over-, under-estimators for products $x_i x_j$ where $x \in [-1, 1]$ (McCormick's envelope):



- Convex relaxation is not the tightest possible, but it can be constructed automatically

Summary

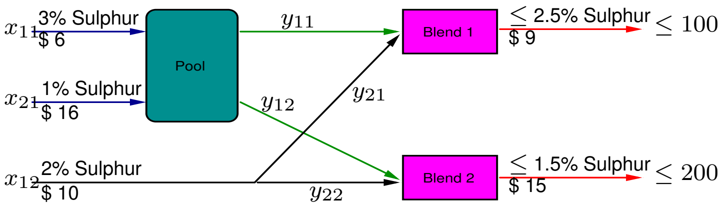
Original MINLP	Standard Form	Convex Relaxation
$\min_x f(x)$ $g(x) \leq 0$ $x^L \leq x \leq x^U$	$\min w_1$ $Aw = b$ $w_i = w_j w_k \ \forall (i,j,k) \in \mathcal{T}_{blt}$ $w_i = \frac{w_j}{w_k} \ \forall (i,j,k) \in \mathcal{T}_{lft}$ $w_i = h_{ij}(w_j) \ \forall (i,j) \in \mathcal{T}_{uf}$ $w^L \leq w \leq w^U$	$\min w_1$ $Aw = b$ McCormick's relaxation Secant relaxation $w^L \leq w \leq w^U$

Some variables may be integral

- Easier to perform symbolic algorithms
- Linearizes nonlinear terms
- Adds linearizing variables and defining constraints

Each defining constraint replaced by convex under- and concave over-estimators

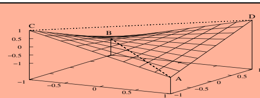
Eg: conv. rel. of pooling problem



$$\left\{\begin{array}{ll} \min\limits_{x,y,p} & 6x_{11} + 16x_{21} + 10x_{12} - \\ & \quad -9(y_{11} + y_{21}) - 15(y_{12} + y_{22}) \\ \text{s.t.} & x_{11} + x_{21} - y_{11} - y_{12} = 0 \text{ linear} \\ & x_{12} - y_{21} - y_{22} = 0 \text{ linear} \\ & y_{11} + y_{21} \leq 100 \text{ linear} \\ & y_{12} + y_{22} \leq 200 \text{ linear} \\ & 3x_{11} + x_{21} - p(y_{11} + y_{12}) = 0 \\ & py_{11} + 2y_{21} \leq 2.5(y_{11} + y_{21}) \\ & py_{12} + 2y_{22} \leq 1.5(y_{12} + y_{22}) \end{array}\right.$$

$$\left\{\begin{array}{ll} \min & \text{cost} \\ \text{s.t.} & \text{linear constraints} \\ & 3x_{11} + x_{21} - w_1 = 0 \\ & w_3 + 2y_{21} \leq 2.5(y_{11} + y_{21}) \\ & w_4 + 2y_{22} \leq 1.5(y_{12} + y_{22}) \\ & w_2 = y_{11} + y_{12} \\ & w_1 = pw_2 \\ & w_3 = py_{11} \\ & w_4 = py_{12} \end{array}\right.$$

Replace nonconvex constr. $w = uv$ by McCormick's envelopes:
$w \geq \max\{u^L v + v^L u - u^L v^L, u^U v + v^U u - u^U v^U\}$,
$w \leq \min\{u^U v + v^L u - u^U v^L, u^L v + v^U u - u^L v^U\}$



Variable ranges

- Crucial property for sBB convergence: **convex relaxation tightens as variable range widths decrease**
- convex/concave under/over-estimator constraints are (convex) functions of x^L, x^U
- it makes sense to **tighten** x^L, x^U at the sBB root node (trading off speed for efficiency) and at each other node (trading off efficiency for speed)

OBBT and FBBT

- In sBB we need to tighten variable bounds at each node
- Two methods: Optimization Based Bounds Tightening (OBBT) and Feasibility Based Bounds Tightening (FBBT)
- OBBT: for each variable x in P compute $\min$ and $\max\{x \mid \text{conv. rel. constr.}\}$, see e.g. [Caprara et al., MP 2009]
- FBBT:
  propagation of intervals up and down constraint expression trees, with tightening at the root node

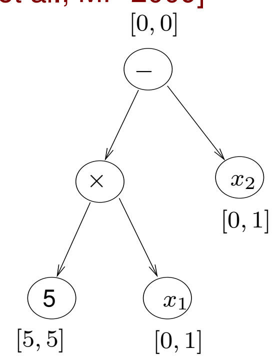
  Example: $5x_1 - x_2 = 0$.

  Up: $\otimes$:$[5,5] \times [0,1] = [0,5]$; $\ominus$:$[0,5] - [0,1] = [-1,5]$.

  Root node tightening: $[-1,5] \cap [0,0] = [0,0]$.

  Downwards: $\otimes$:$[0,0] + [0,1] = [0,1]$;

  x_1:$[0,1]/[5,5] = [0, \frac{1}{5}]$



- Iterating (up/tighten/down) k times yields $[0, \frac{1}{5^{2k-1}}]$

Quadratic problems

- All nonlinear terms are quadratic monomials
- Aim to reduce gap betwen the problem and its convex relaxation
- $\Rightarrow$ replace quadratic terms with suitable linear constraints (fewer nonlinear terms to relax)
- Can be obtained by considering linear relations (called **reduced RLT constraints**) between original and linearizing variables

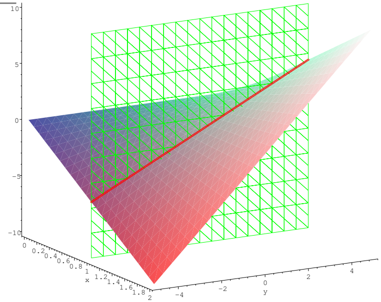
Reduced RLT Constraints I

- For each $k \leq n$, let $w_k = (w_{k1}, \ldots, w_{kn})$
- Multiply $Ax = b$ by each x_k, substitute linearizing variables w_k, get **reduced RLT constraint system** (RRCS)

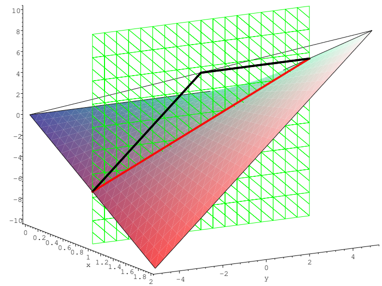
$$\forall k \leq n \ (Aw_k = bx_k)$$

- $\forall\ i, k \leq n$ define $z_{ki} = w_{ki} - x_i x_k$, let $z_k = (z_{k1}, \ldots, z_{kn})$
- Substitute $b = Ax$ in RRCS, get $\forall k \leq n(A(w_k - x_k x) = 0)$, i.e. $\forall k \leq n(Az_k = 0)$. Let B, N be the sets of basic and nonbasic variables of this system
- Setting $z_{ki} = 0$ for each nonbasic variable implies that the RRCS is satisfied $\Rightarrow$ It suffices to enforce quadratic constraints $w_{ki} = x_i x_k$ for $(i, k) \in N$ (replace those for $(i, k) \in B$ with the linear RRCS)

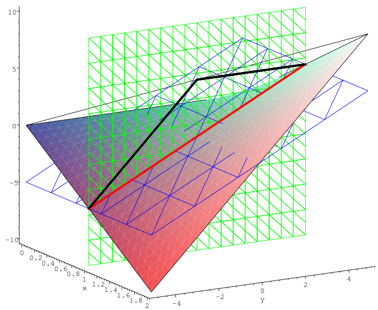
Reduced RLT Constraints II



$F(P) = \{(x,y,w) \mid w = xy \wedge x = 1\}$



McCormick's rel.



rRLT constraint:
multiply $x = 1$ by y, get $xy = y$,
replace xy by w, get $w = y$
$F(P)$ described *linearly*

Reduced RLT Constraints III

- If $|E| = \frac{1}{2}n(n+1)$ (all possible quadratic terms), get $|B|$ fewer quadratic terms in reformulation
- Otherwise, judicious choice of multiplier variable set $\{x_k \mid k \in K\}$ and multiplied linear equation constraint subsystem must be performed.

Citations

- Sherali, Alameddine, *A new reformulation-linearization technique for bilinear programming problems*, JOGO, 1991
- Smith, Pantelides, *A symbolic reformulation/spatial branch-and-bound algorithm for the global optimisation of nonconvex MINLPs*, Comp. Chem. Eng. 1999
- Liberti, *Reduction Constraints for the Global Optimization of NLPs*, ITOR, 2004
- Liberti, *Linearity embedded in nonconvex programs*, JOGO, 2005
- Liberti, Pantelides, *An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms*, JOGO, 2006
- Belotti, Liberti et al., *Branching and bounds tightening techniques for nonconvex MINLP*, Opt. Meth. Softw., 2009
- Sherali, Dalkiran, Liberti, *Reduced RLT representations for nonconvex polynomial programming problems*, JOGO (to appear)

The end