ROSE:
Reformulation/Optimization Software Engine

Sonia Cafieri

LIX, École Polytechnique

joint work with: Leo Liberti, Fabien Tarissan, LIX

ARS Workshop, Oct 31st 2008

Outline

Outline

ROSE: aim and features

ROSE

=

Reformulation/Optimization Software Engine

is a software framework for reformulating and solving Mathematical Programming problems.

Main aim: to provide reformulations of mathematical programs of various types **automatically**.

Specific reformulations are carried out in the form of pre-processing steps by LP/MILP optimization solvers, but there is no software framework able to carry out reformulations in a systematic way.

ROSE: aim and features

- It implements *reformulation solvers*, working towards analysing or changing a problem structure and *numerical solvers*, working towards finding a solution.
  Currently, it is more *focused on reformulation* than optimization.
- Mathematical programs can be reformulated according to several algorithms; the result can be used by other optimization codes.
- It can parse a mathematical program to a well-defined data structure, involving trees used to represent mathematical expressions.
- A separate library called *Ev3* is used to handle expression trees.
- A direct user interface and an AMPL interface are available. ROSE can be used stand-alone as well as an AMPL solver (reformulated problems can be output in AMPL format).

ROSE: aim and features

- It implements *reformulation solvers*, working towards analysing or changing a problem structure and *numerical solvers*, working towards finding a solution.
  Currently, it is more *focused on reformulation* than optimization.
- Mathematical programs can be reformulated according to several algorithms; the result can be used by other optimization codes.
- It can parse a mathematical program to a well-defined data structure, involving trees used to represent mathematical expressions.
- A separate library called *Ev3* is used to handle expression trees.
- A direct user interface and an AMPL interface are available. ROSE can be used stand-alone as well as an AMPL solver (reformulated problems can be output in AMPL format).

ROSE: aim and features

- It implements *reformulation solvers*, working towards analysing or changing a problem structure and *numerical solvers*, working towards finding a solution.
  Currently, it is more *focused on reformulation* than optimization.
- Mathematical programs can be reformulated according to several algorithms; the result can be used by other optimization codes.
- It can parse a mathematical program to a well-defined data structure, involving trees used to represent mathematical expressions.
- A separate library called *Ev3* is used to handle expression trees.
- A direct user interface and an AMPL interface are available. ROSE can be used stand-alone as well as an AMPL solver (reformulated problems can be output in AMPL format).

ROSE: aim and features

- It implements *reformulation solvers*, working towards analysing or changing a problem structure and *numerical solvers*, working towards finding a solution.
  Currently, it is more *focused on reformulation* than optimization.
- Mathematical programs can be reformulated according to several algorithms; the result can be used by other optimization codes.
- It can parse a mathematical program to a well-defined data structure, involving trees used to represent mathematical expressions.
- A separate library called *Ev3* is used to handle expression trees.
- A direct user interface and an AMPL interface are available. ROSE can be used stand-alone as well as an AMPL solver (reformulated problems can be output in AMPL format).

ROSE: aim and features

- It implements *reformulation solvers*, working towards analysing or changing a problem structure and *numerical solvers*, working towards finding a solution.
  Currently, it is more *focused on reformulation* than optimization.
- Mathematical programs can be reformulated according to several algorithms; the result can be used by other optimization codes.
- It can parse a mathematical program to a well-defined data structure, involving trees used to represent mathematical expressions.
- A separate library called *Ev3* is used to handle expression trees.
- A direct user interface and an AMPL interface are available. ROSE can be used stand-alone as well as an AMPL solver (reformulated problems can be output in AMPL format).

ROSE: people

People working on ROSE:

- Leo Liberti (*LIX*)
- Sonia Cafieri (*LIX*)
- Fabien Tarissan (*LIX*)
- Jordan Ninin (*ENSEEIHT, Toulouse*)
- Pete Janes (*Australian National University*)

Outline

What is currently implemented in ROSE?

Reformulators

- convexification/approximation/...
- data analysis/copy/print
- data format translation

Numerical Solvers

- native solvers
- wrappers to external solvers

Outline

Current status – Reformulators

- `solver_Rprodbincont` product of binary and continuous variables reformulator
- `solver_Rsmith` Smith standard form reformulator
- `solver_Rconvexifier` Smith convexifier
- `solver_RQuarticConvex` convexifier for quartic terms
- `solver_Rsymmgroup` MINLP to DAG reformulator, computes the colours to be given to nodes
- `solver_Rcopy` copier (for later reformulations)
- `solver_Rprint` printer (identity reformulation)
- `solver_Rprintmod` printer in AMPL flat form
- `solver_Rprintdat` printer of AMPL files .mod and .dat for LP
- `solver_Rcdd` translator to the input format for CDD software
- `solver_Rporta` translator to the input format for PORTA software
- `solver_Rvinci` translator to the input format for VINCI software

Outline

Current status – Numerical Solvers

- solver_vns VNS solver for nonconvex NLPs
- solver_glpk wrapper for GLPK solver for LPs
- solver_snopt6 wrapper for SNOPT solver for NLPs
- solver_ipopt wrapper for IPOPT solver for NLPs (work in progress)

- solver_limitedbranch branch and bound without bound for MINLPs (it solves an NLP at each node, then picks an integer variable with fractional value, branches by fixing, and loops.)
- solver_localbranch uses vns as a local solver, setting $k = k_{max}$ at each iteration
- solver_tabu inserts a nonconvex spherical constraint around each local solution

Outline

A closer look at some reformulator

ProdBinCont

Given: $v_i v_j$, v_i binary variable and v_j continuous variable with $L_j \leq v_j \leq U_j$.
Basic symbolic reformulation algorithm:

- add a continuous variable w_{ij}
- replace $v_i v_j$ by w_{ij}
- add the constraints:

$$\begin{aligned} w_{ij} &\leq U_j v_i \\ w_{ij} &\geq L_j v_i \\ w_{ij} &\leq v_j - (1 - v_i) L_j \\ w_{ij} &\geq v_j - (1 - v_i) U_j \end{aligned} .$$

A closer look at some reformulator

Convexifier

Basic symbolic reformulation algorithm:

- replace each nonlinear term by an added variable w
- add a defining constraint "$w = \textit{nonlinear term}$" to the problem
- replace each defining constraint by a convex relaxation.

Nonlinear terms:

- $x_i x_j$,
- x_j^{2k} for any $k \in \mathbb{N}$,
- x_j^{2k+1} for any $k \in \mathbb{N}$,
- x_i / x_j.

A closer look at some reformulator

Convexifier

Example: bilinear term $x_i x_j$

- replace $x_i x_j$ by w_{ij}
- add a defining constraint $w_{ij} = x_i x_j$ to the problem
- replace the defining constraint by McCormick's envelope:

$$\begin{aligned} w_{ij} &\geq x_i^L x_j + x_j^L x_i - x_i^L x_j^L \\ w_{ij} &\geq x_i^U x_j + x_j^U x_i - x_i^U x_j^U \\ w_{ij} &\leq x_i^L x_j + x_j^U x_i - x_i^L x_j^U \\ w_{ij} &\leq x_i^U x_j + x_j^L x_i - x_i^U x_j^L . \end{aligned}$$

A closer look at some reformulator

Quartic Convexifier

The same algorithm as for the convexifier, specialized for quartic terms:

$$x_1 x_2 x_3 x_4, \;\; x_1 x_2 x_3^2, \;\; x_1 x_2^3, \;\; x_1^2 x_2^2.$$

For quadrilinear terms, different ways of combining terms

$$((x_1 x_2) x_3) x_4, \;\; (x_1 x_2)(x_3 x_4), \;\; (x_1 x_2 x_3) x_4$$

due to the associativity of the product, are considered and in turn different convex relaxations (exploiting the biliner envelopes thrice or the bilinear and the trilinear envelopes).

Outline

Problem & Solver classes

The architecture is mainly based on two classes: `Problem` and `Solver`.

- The `Problem` class has methods for reading in a problem, access/modify the problem description, perform various reformulations to do with adding/deleting variables and constraints, evaluate the problem expressions and their first and second derivatives at a given point, and test for feasibility of a given point in the problem.
- The `Solver` class is a virtual class that serves as interface for various solvers.
  Implementations of this class may be *numerical solvers* or *reformulation solvers*.

Outline

Problem representation

ROSE represents optimization problems in their *flat form* representation: variables, objective functions and constraints are arranged in simple linear lists.

- `struct Variable`, storing informations on decision variables (ID, name, lower and upper bound,...)
- `struct Objective`, storing informations on objective functions (ID, expression tree, expression tree of the nonlinear part, opt direction, prime and second order partial derivatives, ...)
- `struct Constraint`, storing informations on constraints (ID, expression tree, expression tree of the nonlinear part, lower and upper bound, prime and second order partial derivatives, ...)

Input problem example

```
variables = -1 < x < 1,
            -2 < y < 3;
objfun = [ x*y + 2*x^2 ];
constraints = [ 2 < x + y < PlusInfinity ];
startingpoint = 0, 0;
```

Reformulator selection

Choose the convexifier reformulator

Run ROSE:

```
rose -s Rconvexifier input/bilin-convex.ros
```

Output

output file Rconvexifier_out.ros:

```
# ROSE problem: bilin-convex
# Problem has 5 variables and 12 constraints
# Variables:

variables = -1 < x < 1 / Continuous,
-2 < y < 3 / Continuous,
0 < w3 < 1 / Continuous,
-3 < w4 < 3 / Continuous,
-3 < w5 < 5 / Continuous;

# Objective Function:
objfun = min [ w_5 ];

# Constraints:
constraints = [ 2 < (x_1)+(y_2) < 1e+30 ],
[ 0 < (2*w_3)+(w_4)+(-1*w5_5) < 0 ],
[ -1 < (2*x_1)+(w3_3) < 1e+30 ],
[ -1 < (-2*x_1)+(w3_3) < 1e+30 ],
[ -0.25 < (x_1)+(w3_3) < 1e+30 ],
[ -0.25 < (-1*x_1)+(w3_3) < 1e+30 ],
[ -0 < w3_3 < 1e+30 ],
[ -1e+30 < w3_3 < 1 ],
[ -2 < (2*x_1)+(y_2)+(w4_4) < 1e+30 ],
[ -3 < (-3*x_1)+(-1*y_2)+(w4_4) < 1e+30 ],
[ -1e+30 < (-3*x_1)+(y_2)+(w4_4) < 3 ],
[ -1e+30 < (2*x_1)+(-1*y_2)+(w4_4) < 2 ];

# Starting Point:
startingpoint = 0, 0, 0, 0, 0;

# end of problem bilin-convex
```

Future perspective

- Adding new reformulators.
- Unifying the convexifiers.
- Extensive testing.
- Contributions to the further development are welcome!

Future perspective

- Adding new reformulators.
- Unifying the convexifiers.
- Extensive testing.
- Contributions to the further development are welcome!

Future perspective

- Adding new reformulators.
- Unifying the convexifiers.
- Extensive testing.
- Contributions to the further development are welcome!

Future perspective

- Adding new reformulators.
- Unifying the convexifiers.
- Extensive testing.
- Contributions to the further development are welcome!

Future perspective

- Adding new reformulators.
- Unifying the convexifiers.
- Extensive testing.
- Contributions to the further development are welcome!