Advances in Automated Conversion of Optimization Problems

Robert Fourer, Gleb Belov, Filipe Brandão

[4er,gleb,fdabrandao]@ampl.com

AMPL Optimization Inc.
www.ampl.com — +1 773-336-AMPL

Workshop on Recent Advances in Optimization

The Fields Institute
Toronto, 11-12 October 2023

Advances in Automated Conversion of Optimization Problems

We take it for granted that an optimization package accepts both minimization and maximization problems, recognizes them as equivalent, and converts all minimizations to maximizations (or vice-versa) before solving. This is only the very simplest example of the conversions that large-scale optimization relies on. In the past decade, the range of expressions recognized by modeling languages and solvers has been progressively extended in ways that make conversion possibilities ever more numerous and complex. This presentation describes a range of challenges and accomplishments in detection of formulations that solvers can handle, and in transformation to forms that solvers require. Examples from integer, logic, and conic programming lead to some general recommendations for design and implementation.

Outline

Motivation

Principles

- How and where the conversion happens
- How problems are represented

  + a variety of examples in current software

AMPL "MP" interface

- Formulating models more like you think about them
- Extensions for MIP solvers
- Issues: implementation, solver efficiency, solver tolerances

Still-challenging cases

- Convex functions
- Second-order cones

Conversions We Take for Granted

Minimize to Maximize

- Change sign, solve maximization, change sign back

$\geq, \leq, =$ *to standard form*

- Add slack variables

Linear expressions to coefficient lists

- Distribute a constant over a sum of variables
- Merge appearances of the same variable

. . . what's the big deal?

Typical MIP User Complaint

Thank you so much for replying.
Let me show my "if-then" constraint in a more clear way as follows:

```
set veh := {1..16 by 1};

param veh_ind {veh};
param theory_time {veh};
param UP := 400000;

var in_lane_veh {veh} integer >=1, <=2;
var in_in_time {veh} >=0, <=UP;
```

Note that "in_lane_veh {veh}" are integer variables which equal 1 or 2,
and "in_in_time {veh}" are continuous variables.

```
subject to IfConstr {i in 1..card(veh)-1, j in i+1..card(veh):
  veh_ind[i] = veh_ind[j] and theory_time[i] <= theory_time[j]}:

    in_lane_veh[i] = in_lane_veh[j] ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

When I run my program, there appears the following statement:

```
CPLEX 20.1.0.0: logical constraint _slogcon[1] is not an indicator constraint.
```

Typical Reply

To reformulate this model in a way that your MIP solver would accept,
you could define some more binary variables,

```
var in_lane_same {veh,veh} binary;
```

with the idea that in_lane_same[i,j] should be 1 if and only if in_lane_veh[i] = in_lane_veh[j].
Then the desired relation could be written as two constraints:

```
in_lane_veh[i] = in_lane_veh[j] ==> in_lane_same[i,j] = 1
in_lane_same[i,j] = 1 ==> in_in_time[j] >= in_in_time[i] + l_veh/V;
```

The second one is an indicator constraint, but you would just need
to replace the first one by equivalent linear constraints.

Given that in_lane_veh can only be either 1 or 2, those constraints could be

```
in_lane_same[i,j] >= 3 - in_lane_veh[i] - in_lane_veh[j]
in_lane_same[i,j] >= in_lane_veh[i] + in_lane_veh[j] - 3
```

Typical Nonlinear User Complaint

So I tried out gurobi with the two commands I mentioned in my previous email,
and I receive the message

```
Gurobi 9.0.2: Gurobi can't handle nonquadratic nonlinear constraints.
```

I went over the constraints, and it seems to me
the only constraint that is nonquadratic nonlinear is

```
subject to A2 {t in 2..card(POS), i in PATIENTS}:
   sum {a in DONORS, b in PATIENTS, c in PATIENTS: ceil(a/2) = c}
      x[b,t] * x[c,t-1] * y[a,b] = 2 * x[i,t];
```

where x and y are binary variables.

Is this now sufficient for gurobi to solve if I only linearize one of the term
on the LHS of this constraint (e.g. x[b, t]), while keeping the other two terms the same?

Typical Reply

*You are right, A2 has a cubic term x[b,t] * x[c,t-1] * y[a,b] that you will have to transform before you can get Gurobi to accept it.*

You can transform to quadratic by picking two of the three variables and replacing their product by a new variable. For example, if you define a new binary variable z[b,c,t] to replace x[b,t] * x[c,t-1], you can write

```
var z {t in 2..card(POS), b in PATIENTS, c in PATIENTS} binary;
subject to zDefn {t in 2..card(POS), b in PATIENTS, c in PATIENTS}:
   z[b,c,t] = x[b,t] * x[c,t-1];
```

Then write your constraint A2 as z[b,c,t] * y[a,b] = 2 * x[i,t]. There are two other possibilities, corresponding to the two other ways you can pick two of the three variables.

You can also linearize the cubic term directly. In that case, you would define a new binary variable z[a,b,c,t] to replace x[b,t] * x[c,t-1] * y[a,b], and you would add the following four constraints:

```
z[a,b,c,t] >= x[b,t] + x[c,t-1] + y[a,b] - 2
z[a,b,c,t] <= x[b,t]
z[a,b,c,t] <= x[c,t-1]
z[a,b,c,t] <= y[a,b]
```

Principles

Phases of automated conversion

- Detection and transformation
- Solver-dependent and solver-independent

Stages where the conversion happens

- User, modeling language, interface, solver

Problem representations

- Lists, Graphs

Examples of conversion in practice

- Libraries, file formats, transformations

Phases of Automated Conversion

Detection (solver-independent)

- Identify objectives and constraints
  that admit some desirable form of conversion

Transformation (solver-independent)

- Convert to equivalent forms that solvers may handle

Transformation (solver-dependent)

- Convert to specific forms required by solver APIs

Combine these?

- Yes, to support one solver most efficiently
- No, to support many solvers in a consistent way

Stages Where Conversion Happens

Not automated

- User reformulation "by hand"

Automated: Symbolic model

- Automated conversion of a modeling language representation

Automated: Explicit optimization problem

- Conversion within a modeling system
- ***Conversion by the solver interface***
- Conversion inside the solver

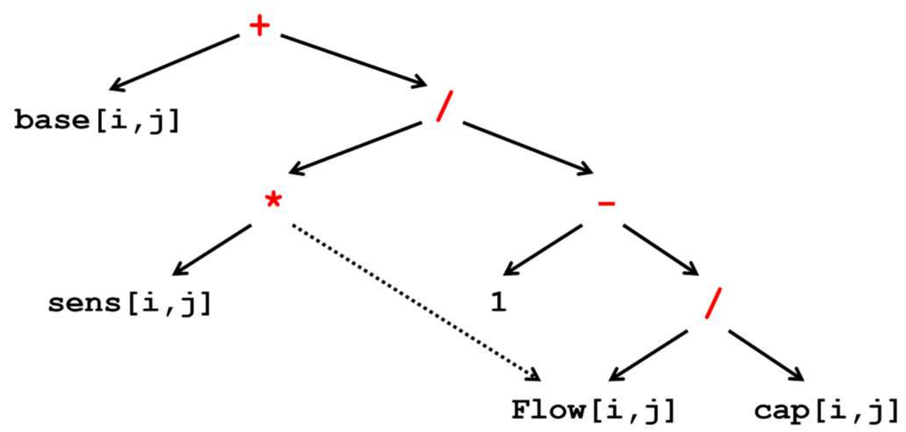
Problem Representations

Lists

- Linear coefficients
- Quadratic coefficients

Directed acyclic graphs

- Concise form of trees
- Internal nodes: operators, functions
- Terminal nodes: variables, constants



Detection: isQuadr()

```
boolean isQuadr (Node);

case of Node {

 PLUS:
 MINUS: return( isQuadr(Node.left) and isQuadr(Node.right) );

 TIMES: return( isLinear(Node.left) and isLinear(Node.right) or
                isQuadr(Node.left) and isConst(Node.right)   or
                isConst(Node.left) and isQuadr(Node.right) );

 POWER: return( isLinear(Node.left) and
                isConst(Node.right) and value(Node.right) == 2 );

 VAR:   return( TRUE );

 CONST: return( TRUE );

}
```

. . . to detect, test `isQuadratic(root)`

Examples *(library & file format)*

AMPL-solver library

- C interface library and *.nl* file format
- Coefficient lists for linear expressions
  + graphs for nonlinear & logical expressions
- Some conversions: quadratic, indicator, piecewise-linear
  - David M. Gay, "Writing **.nl** Files." *https://ampl.github.io/nlwrite.pdf*

MathOptInterface (JuMP)

- Julia interface library and JSON-based file format
- $f_i(x) \in S_i$ representations for linear, quadratic; *many conic sets*
- Systematic conversion between representations
  - Benoît Legat, Oscar Dowson, Joaquim Dias Garcia, Miles Lubin, "MathOptInterface: A Data Structure for Mathematical Optimization Problems." *INFORMS Journal on Computing* 34 (2022) 672–689.

Examples *(transformation)*

MiniZinc

- Modeling language for Constraint Programming
- Linearization of diverse logical conditions to support MIP solvers
  - Gleb Belov, Peter J. Stuckey, Guido Tack, Mark Wallace, "Improved Linearization of Constraint Programming Models." CP 2016: International Conference on Principles and Practice of Constraint Programming. *Lecture Notes in Computer Science* **9892** (Springer, 2016) 49–65.

gurobipy

- Python modeling language & interface to Gurobi solver
- Linear, quadratic + "general" constraints
  - max, abs, or, norm, indicator, piecewise-linear
  - exp, log, power, sin, cos . . .
- Transformation to linear-quadratic MIPs
  - *https://www.gurobi.com/documentation/current/refman/constraints.html*

Example *(library & transformation)*

MP library

- C++ library for building efficient, configurable solver interfaces
- High-performance *.nl* file reader
- Coefficient lists + expression graphs
- Extensive toolset for detection and transformation
  - “MP Library.” *https://amplmp.readthedocs.io/*
  - Gleb Belov, “How to Hook Your Solver to AMPL MP.” *https://mp.ampl.com/howto.html, https://github.com/ampl/mp/tree/develop/solvers/visitor*

Writing (MIP) **Models More Like You Think About Them**

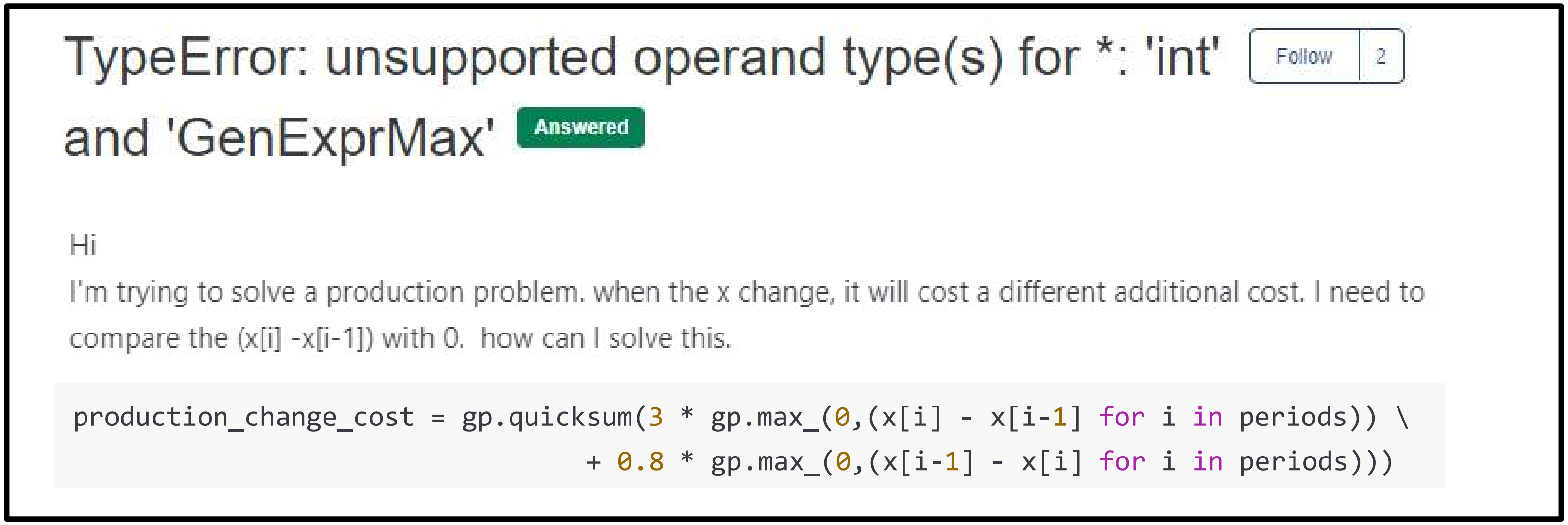
General context

- AMPL has logical and “not linear” expressions
- Previous ASL interface had very limited support for these
- New interfaces, built with MP,
  allow these expressions to be used and *combined* generally

Gurobi context

- AMPL should support Gurobi’s general constraints
- Existing gurobipy offers only limited generality
- New interfaces, built with MP,
  convert much more general expressions to work with Gurobi

Maximum in gurobipy



Maximum in gurobipy *(reply)*

General constraints are meant to be used to define single constraints. It is not possible to use these constructs in other expressions, i.e., it is not possible to use gp.max_ in a more complex constraint other than y = gp.max_.

Moreover, as described in the [documentation of the addGenConstrMax method](), gp.max_ only accepts single variables as inputs. Thus, it is not possible to pass expressions **x[i] -x[i-1]**. To achieve what you want, you have to introduce additional auxiliary variables **aux[i] = x[i] -x[i-1]** and additional equality constraints z1 = gp.max_ and z2 = gp.max_

```
aux1 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar1")
aux2 = mod.addVars(periods, lb=-GRB.INFINITY, name="auxvar2")
# are you sure that i-1 does not lead to a wrong key access?
m.addConstrs((aux1[i] = x[i]-x[i-1| for i in periods), name = "auxconstr1")
m.addConstrs((aux2[i] = x[i-1]-x[i| for i in periods), name = "auxconstr2")
z1 = m.addVar(lb = -GRB.INFINITY, name="z1")
z2 = m.addVar(lb = -GRB.INFINITY, name="z2")
m.addConstr(z1 = gp.max_(0,aux1),name="maxconstr1")
m.addConstr(z2 = gp.max_(0,aux2),name="maxconstr2")
[...]
production_change_cost = gp.quicksum(3 * z1 + 0.8 * z2)
```

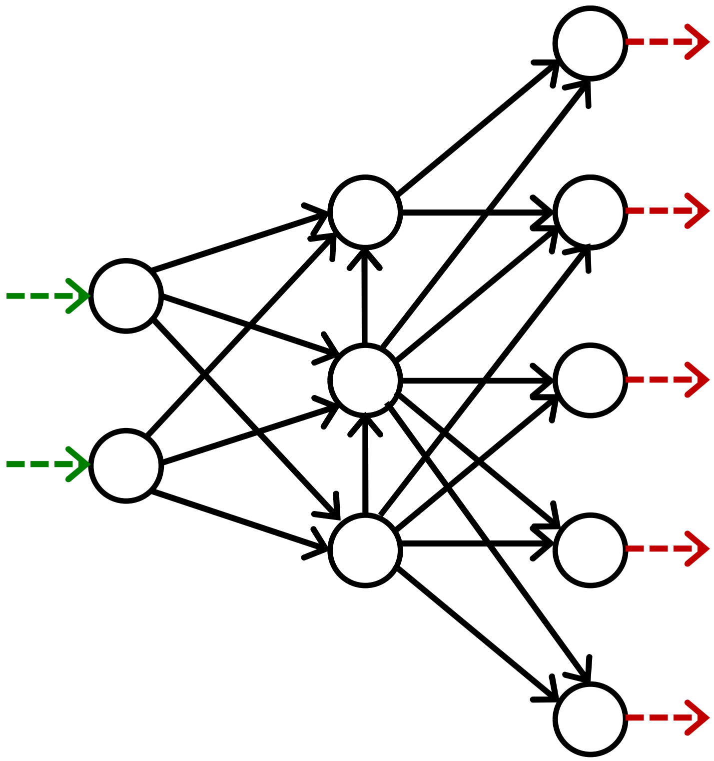
Example:
Multi-Product Network Flow

Motivation

- Ship products efficiently to meet demands

Context

- a transportation network
  - nodes ◯ representing cities
  - arcs ⟶ representing roads
- supplies ---> at nodes
- demands ---> at nodes
- capacities on arcs
- shipping costs on arcs



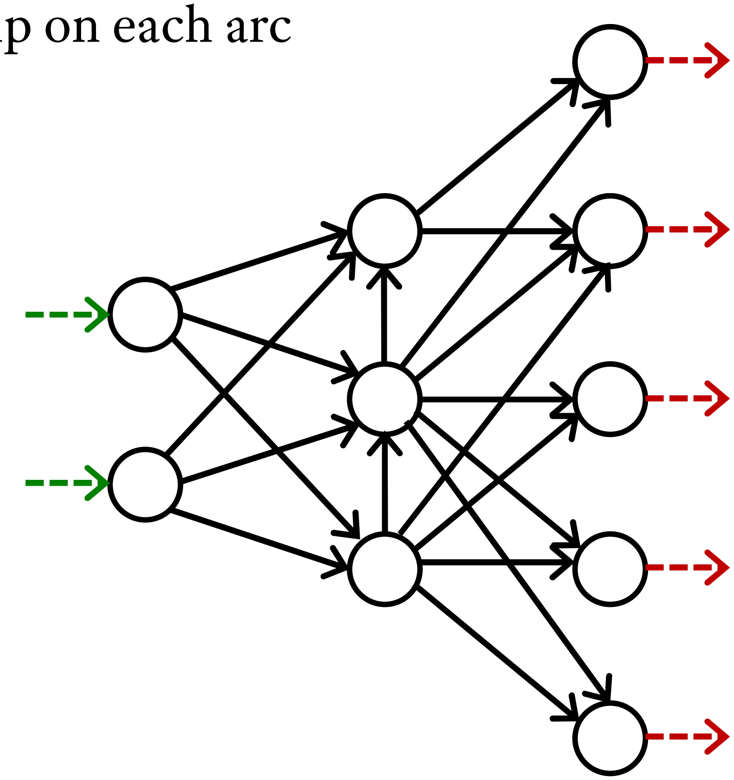
Example:
Multi-Product Network Flow

Decide

- how much of each product to ship on each arc

So that

- shipping costs are kept low
- shipments on each arc respect capacity of the arc
- supplies, demands, and shipments are in balance at each node



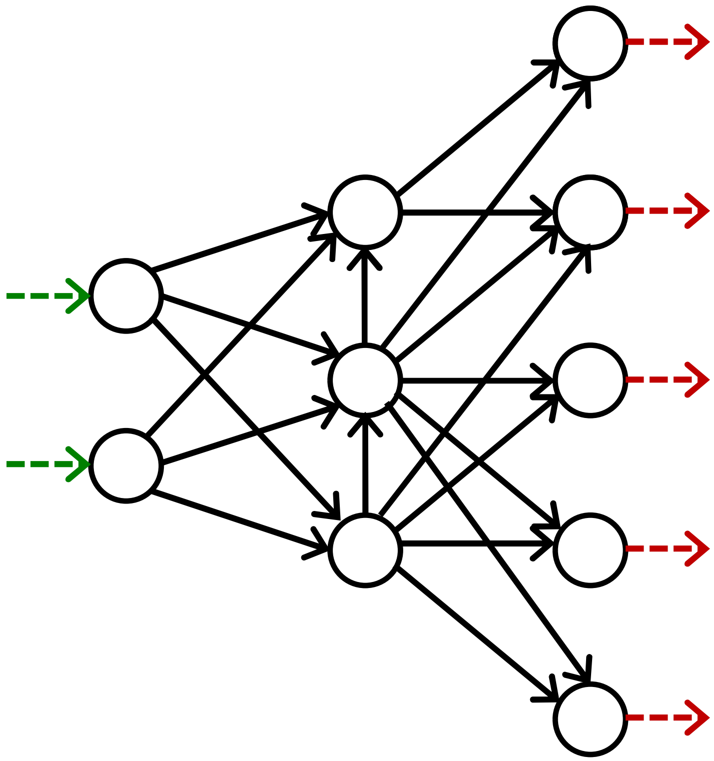
Example with complications:
Multi-Product Network Flow

Decide also

- whether to use each arc

So that

- variable plus fixed
  shipping costs are kept low
- shipments are not too small
- not too many arcs are used



Formulation *(data)*

Given

- P set of products
- N set of network nodes
- $A \subseteq N \times N$ set of arcs connecting nodes

and

- u_{ij} capacity of arc from i to j, for each $(i, j) \in A$
- s_{pj} supply/demand of product p at node j, for each $p \in P$, $j \in N$
  > 0 implies supply, < 0 implies demand
- c_{pij} cost per unit to ship product p on arc (i, j),
  for each $p \in P$, $(i, j) \in A$
- d_{ij} fixed cost for using the arc from i to j, for each $(i, j) \in A$
- m smallest total shipments on any arc that is used
- n largest number of arcs that may be used

Linearized Formulation *(variables, objective)*

Determine

X_{pij} amount of commodity p to be shipped on arc (i,j),
for each $p \in P$, $(i,j) \in A$

Y_{ij} 1 if any amount is shipped from node i to node j,
0 otherwise, for each $(i,j) \in A$

to minimize

$$\sum_{p\in P} \sum_{(i,j)\in A} c_{pij}\, X_{pij} + \sum_{(i,j)\in A} d_{ij}\, Y_{ij}$$

total cost of shipments

Linearized Formulation *(constraints)*

Subject to

$\sum_{p \in P} X_{pij} \leq u_{ij} Y_{ij}$, for all $(i,j) \in A$

when the arc from node i to node j is used for shipping,
total shipments must not exceed capacity, and Y_{ij} must be 1

$\sum_{p \in P} X_{pij} \geq m Y_{ij}$, for all $(i,j) \in A$

when the arc from node i to node j is used for shipping,
total shipments from i to j must be at least m

$\sum_{(i,j) \in A} X_{pij} + s_{pj} = \sum_{(j,i) \in A} X_{pji}$, for all $p \in P$, $j \in N$

shipments in plus supply/demand must equal shipments out

$\sum_{(i,j) \in A} Y_{ij} \leq n$

At most n arcs can be used

Multi-Product Flow

Linearized Model in AMPL

Symbolic data, variables, objective

```
set PRODUCTS;
set NODES;

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;

param inflow {PRODUCTS,NODES};
param min_ship >= 0;
param max_arcs >= 0;

param var_cost {PRODUCTS,ARCS} >= 0;
var Flow {PRODUCTS,ARCS} >= 0;

param fix_cost {ARCS} >= 0;
var Use {ARCS} binary;

minimize TotalCost:
   sum {p in PRODUCTS, (i,j) in ARCS} var_cost[p,i,j] * Flow[p,i,j] +
   sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

Multi-Product Flow

Linearized Model in AMPL

Constraints

```
subject to Capacity {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j] * Use[i,j];

subject to Min_Shipment {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[i,j];

subject to Conservation {p in PRODUCTS, j in NODES}:
   sum {(i,j) in ARCS} Flow[p,i,j] + inflow[p,j] =
   sum {(j,i) in ARCS} Flow[p,j,i];

subject to Max_Used:
   sum {(i,j) in ARCS} Use[i,j] <= max_arcs;
```

Formulating

Positive Shipments Incur Fixed Costs

Linearized formulation

```
sum {(i,j) in ARCS} fix_cost[i,j] * Use[i,j];
```

Natural formulation

```
sum {(i,j) in ARCS}
   if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j]
```

Formulating

Shipments Can't Be Too Small

Linearized formulation

```
sum {p in PRODUCTS} Flow[p,i,j] >= min_ship * Use[i,j];
sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j] * Use[i,j];
```

Natural formulation

```
sum {p in PRODUCTS} Flow[p,i,j] = 0 or
min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j]
```

Formulating

Can’t Use Too Many Arcs

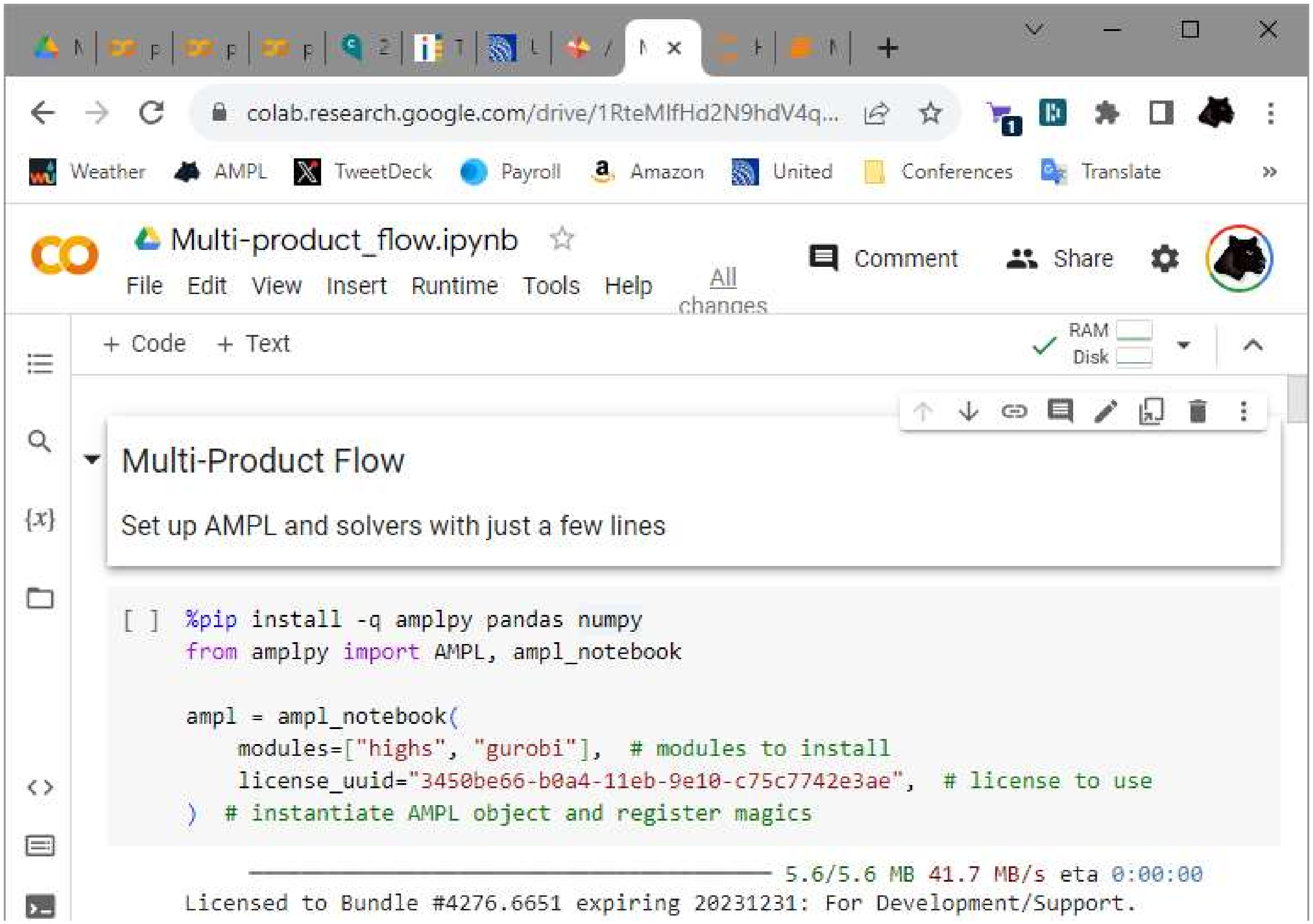
Linearized formulation

```
sum {(i,j) in ARCS} Use[i,j] <= max_arcs;
```

Natural formulation

```
atmost max_arcs {(i,j) in ARCS}
   (sum {p in PRODUCTS} Flow[p,i,j] > 0);
```

Solving: AMPL & Gurobi on Google Colab



https://colab.research.google.com/drive/1RteMlfHd2N9hdV4q7luEf5X9ElgxeYR0?usp=sharing

Solving

AMPL Model in Notebook Cell

colab.research.google.com/drive/1RteMIfHd2N9hdV4q...

+ Code + Text All changes saved

AMPL model

Using new logical operators (no binary variables)

```
%%ampl_eval
set PRODUCTS;
set NODES;

set ARCS within {NODES,NODES};
param capacity {ARCS} >= 0;
```

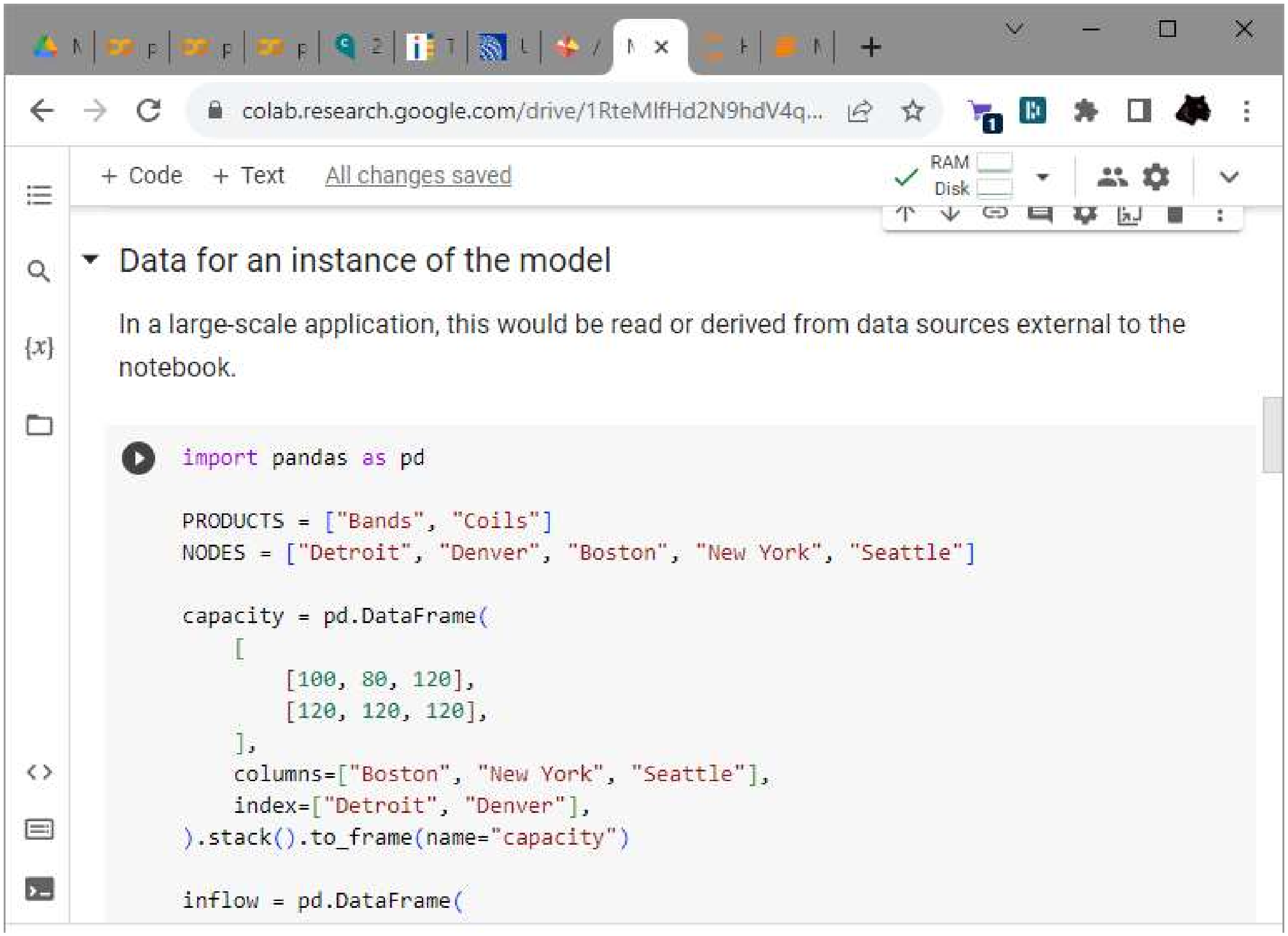
```
subject to Shipment_Limits {(i,j) in ARCS}:
    sum {p in PRODUCTS} Flow[p,i,j] = 0 or
    min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

subject to Conservation {p in PRODUCTS, j in NODES}:
    sum {(i,j) in ARCS} Flow[p,i,j] + inflow[p,j] =
    sum {(j,i) in ARCS} Flow[p,j,i];

subject to Limit_Used:
    atmost max_arcs {(i,j) in ARCS}
        (sum {p in PRODUCTS} Flow[p,i,j] > 0);
```

Solving

Python Data for the Model



Solving

Passing the Data to AMPL

Passing the data to AMPL

```
# product and node sets
ampl.set["PRODUCTS"] = PRODUCTS
ampl.set["NODES"] = NODES

# arc set and capacity
ampl.set_data(capacity, "ARCS")

# inflow
ampl.param["inflow"] = inflow

# min_ship, max_arcs
ampl.param["min_ship"] = min_ship
ampl.param["max_arcs"] = max_arcs

# var_cost, fix_cost
ampl.param["var_cost"] = var_cost
ampl.eval("data; param fix_cost default 75;")
```

Solving

Invoking the Solver

```
[5]  # solve
     ampl.option["solver"] = "gurobi"
     ampl.solve()

     # retrieve solution
     ampl.option["display_1col"] = "0"
     ampl.eval("display Flow;")

     ampl.var["Flow"].to_pandas()
```

```
Gurobi 10.0.3: optimal solution; objective 5900
10 simplex iterations
1 branching nodes

Flow [Bands,*,*] (tr)
:           Denver Detroit    :=
Boston         0      50
'New York'    50       0
Seattle       10       0

 [Coils,*,*] (tr)
:           Denver Detroit    :=
Boston         0      40
'New York'    10      20
Seattle       30       0
```

Supported Extensions and Solvers

Operators and functions

- Conditional: `if-then-else`; `==>`, `<==`, `<==>`
- Logical: `or`, `and`, `not`; `exists`, `forall`
- Piecewise linear: `abs`; `min`, `max`; `<<breakpoints; slopes>>`
- Counting: `count`; `atmost`, `atleast`, `exactly`; `numberof`
- Comparison: `>`, `<`, `!=`; `alldiff`
- Complementarity: `complements`
- Nonlinear: `*`, `/`, `^`; `exp`, `log`; `sin`, `cos`, `tan`; `sinh`, `cosh`, `tanh`
- Set membership: `in`

Expressions and constraints

- High-order polynomials
- Second-order and exponential cones

Extensions for MIP Solvers

Conditional operators

- **if** *constraint* **then** *var-expr1* [**else** *var-expr2*]
- *constraint1* ==> *constraint2* [**else** *constraint3*]
  constraint1 <== *constraint2*
  constraint1 <==> *constraint2*

```
minimize TotalCost:
   sum {j in JOBS, k in MACHINES}
      if MachineForJob[j] = k then cost[j,k];
```

```
subject to Multi_Min_Ship {i in ORIG, j in DEST}:
   sum {p in PROD} Trans[i,j,p] >= 1 ==>
      minload <= sum {p in PROD} Trans[i,j,p] <= limit[i,j];
```

Extensions for MIP Solvers

Logical operators

- *constraint1* **or** *constraint2*
  constraint1 **and** *constraint2*
  not *constraint2*
- **exists** {*indexing*} *constraint-expr*
  forall {*indexing*} *constraint-expr*

```
subject to NoMachineConflicts
      {m1 in 1..nMach, m2 in m1+1..nMach, j in 1..nJobs}:
   Start[m1,j] + duration[m1,j] <= Start[m2,j] or
   Start[m2,j] + duration[m2,j] <= Start[m1,j];
```

```
subj to HostNever {j in BOATS}:
   isH[j] = 1 ==> forall {t in TIMES} H[j,t] = j;
```

Extensions for MIP Solvers

Piecewise-linear functions and operators

- << *breakpoint-list* ; *slope-list* >> *variable*
  << *breakpoint-list* ; *slope-list* >> (*variable* , *zero-point*)
- **abs**(*var-expr*)
  min(*var-expr-list*) **min** {*indexing*} *var-expr*
  max(*var-expr-list*) **max** {*indexing*} *var-expr*

(gurobipy)

```
x = mod.addVars(periods)

production_change_cost = \
  gp.quicksum(3.0 * gp.max_(0,(x[i] - x[i-1] for i in periods)) \
            + 0.8 * gp.max_(0,(x[i-1] - x[i] for i in periods)))
```

(AMPL)

```
var x {0..T} >= 0;

var production_change_cost =
   3.0 * max(0, {i in 1..T} x[i] - x[i-1]) +
   0.8 * max(0, {i in 1..T} x[i-1] - x[i]);
```

Formulating

Extensions for MIP Solvers

Piecewise-linear functions and operators

- << *breakpoint-list* ; *slope-list* >> *variable*
  << *breakpoint-list* ; *slope-list* >> (*variable* , *zero-point*)
- **abs** (*var-expr*)
  min (*var-expr-list*)   **min** { *indexing* } *var-expr*
  max (*var-expr-list*)   **max** { *indexing* } *var-expr*

```
maximize WeightSum:
   sum {t in TRAJ} max {n in NODE} weight[t,n] * Use[n];
```

```
minimize Total_Cost:
   sum {i in ORIG, j in DEST}
      <<{p in 1..npiece[i,j]-1} limit[i,j,p];
        {p in 1..npiece[i,j]} rate[i,j,p]>> Trans[i,j];
```

Extensions for MIP Solvers

Counting operators

- **count** {*indexing*} (*constraint-expr*)
- **atmost** *k* {*indexing*} (*constraint-expr*)
  atleast *k* {*indexing*} (*constraint-expr*)
  exactly *k* {*indexing*} (*constraint-expr*)
- **numberof** *k* **in** (*var-expr-list*)

```
subject to Limit_Used:
  count {(i,j) in ARCS}
    (sum {p in PRODUCTS} Flow[p,i,j] > 0) <= max_arcs;
```

```
subj to CapacityOfMachine {k in MACHINES}:
   numberof k in ({j in JOBS} MachineForJob[j]) <= cap[k];
```

Extensions for MIP Solvers

Comparison operators

- *var-expr1* **!=** *var-expr2*
  var-expr1 **>** *var-expr2*
  var-expr1 **<** *var-expr2*
- **alldiff** (*var-expr-list*)
  alldiff {*indexing*} *var-expr*

```
subj to Different_Colors {(c1,c2) in Neighbors}:
   Color[c1] != Color[c2];
```

```
subject to OnePersonPerPosition:
   alldiff {i in 1..nPeople} Pos[i];
```

Extensions for MIP Solvers

Complementarity operators

- *single-inequality1* **complements** *single-inequality2*
- *double-inequality* **complements** *var-expr*
  var-expr **complements** *double-inequality*

```
subject to Pri_Compl {i in PROD}:
   max(500.0, Price[i]) >= 0 complements
      sum {j in ACT} io[i,j] * Level[j] >= demand[i];
```

```
subject to Lev_Compl {j in ACT}:
   level_min[j] <= Level[j] <= level_max[j] complements
      cost[j] - sum {i in PROD} Price[i] * io[i,j];
```

Extensions for MIP Solvers

Nonlinear expressions and operators

- *var-expr1* ***** *var-expr2*
  var-expr1 **/** *var-expr2*
  var-expr **^** *k*
- **exp**(*var-expr*) **log**(*var-expr*)
  sin(*var-expr*) **cos**(*var-expr*) **tan**(*var-expr*)

```
subj to Eq {i in J} :
   x[i+neq] / (b[i+neq] * sum {j in J} x[j+neq] / b[j+neq]) =
   c[i] * x[i] / (40 * b[i] * sum {j in J} x[j] / b[j]);
```

```
minimize Chichinadze:
     x[1]^2 - 12*x[1] + 11 + 10*cos(pi*x[1]/2)
   + 8*sin(pi*5*x[1]) - exp(-(x[2]-.5)^2/2)/sqrt(5);
```

Extensions for MIP Solvers

Discrete variable domains

- **var** *varname* **{***indexing***} in** *set-expr* **;**

```
var Buy {f in FOODS} in {0,10,30,45,55};
```

```
var Ship {(i,j) in ARCS}
   in {0} union interval[min_ship,capacity[i,j]];
```

```
var Work {j in SCHEDS} integer
   in {0} union interval[least,max {i in SHIFT_LIST[j]} req[i]];
```

Implementation Issues

Is an expression repeated?

- Detect common subexpressions

```
subject to Shipment_Limits {(i,j) in ARCS}:

sum {p in PRODUCTS} Flow[p,i,j] = 0 or
min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];
```

Is there an easy reformulation?

- Yes for min-max, no for max-min

```
minimize Worst_Rank:
  max {i in PEOPLE} sum {j in PROJECTS} rank[i,j] * Assign[i,j];
```

```
maximize Max_Value:
  sum {t in T} max {n in N} weight[t,n] * Value[n];
```

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Flow[i,j] = 0 ==> Use[i,j] = 0 else Use[i,j] = 1;
```

Implementation Issues *(cont'd)*

Does an exact linearization exist?

- Yes if constraint set is "closed"
- No if constraint set is "open"

```
var Flow {ARCS} >= 0;
var Use {ARCS} binary;

subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] >= 0;
```

```
subj to Use_Definition {(i,j) in ARCS}:
   Use[i,j] = 0 ==> Flow[i,j] = 0 else Flow[i,j] > 0;
```

Formulating

Solver Efficiency Issues

Bounds on subexpressions

- Define auxiliary variables that can be bounded

```
var x {1..2} <= 2, >= -2;

minimize Goldstein-Price:
  (1 + (x[1] + x[2] + 1)^2
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + (2*x[1] - 3*x[2])^2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

```
var t1 >= 0, <= 25;   subj to t1def: t1 = (x[1] + x[2] + 1)^2;
var t2 >= 0, <= 100;  subj to t2def: t2 = (2*x[1] - 3*x[2])^2;

minimize Goldstein-Price:
  (1 + t1
    * (19 - 14*x[1] + 3*x[1]^2 - 14*x[2] + 6*x[1]*x[2] + 3*x[2]^2))
* (30 + t2
    * (18 - 32*x[1] + 12*x[1]^2 + 48*x[2] - 36*x[1]*x[2] + 27*x[2]^2));
```

Solver Efficiency Issues *(cont'd)*

Simplification of logic

- Replace an iterated **`exists`** with a **`sum`**

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if exists {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

```
minimize TotalCost: ...
  sum {(i,j) in ARCS}
    if sum {p in PRODUCTS} Flow[p,i,j] > 0 then fix_cost[i,j];
```

Solver Efficiency Issues *(cont'd)*

Creation of common constraint expressions

- Substitute a stronger bound from a constraint

```
subject to Shipment_Limits {(i,j) in ARCS}:
   sum {p in PRODUCTS} Flow[p,i,j] = 0 or
   min_ship <= sum {p in PRODUCTS} Flow[p,i,j] <= capacity[i,j];

minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] > 0
         then fix_cost[i,j];
```

```
minimize TotalCost: ...
   sum {(i,j) in ARCS}
      if sum {p in PRODUCTS} Flow[p,i,j] >= min_ship
         then fix_cost[i,j];
```

. . . consider automating all these improvements

Formulating

Solver Tolerance Issues

Tolerances are applied to linearized expressions

- AMPL might not compute same values as solvers

```
var x {1..2} >=0, <=100;

maximize Total:
   if x[1] <= 4.9999999 and x[2] >= 5.0000001
      then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.9999998

ampl: display x;
1  4.9999999
2  4.9999999

ampl: display Total;
Total = 0
```

Formulating

Solver Tolerance Issues *(cont'd)*

Warning added (needs work)

```
var x {1..2} >=0, <=100;

maximize Total:
   if x[1] <= 4.9999999 and x[2] >= 5.0000001
      then x[1] + x[2] else 0;

subj to con: x[1] = x[2];
```

```
ampl: solve;
Gurobi 10.0.2: optimal solution; objective 9.9999998

------------ WARNINGS ------------
WARNING:  "Solution Check (Idealistic)“
     [ sol:chk:feastol=1e-06, :feastolrel=1e-06, :inttol=1e-05,
       :round='', :prec=‘’ ]
Objective value violations:
  - 1 objective value(s) violated,
        up to 1E+01 (abs)
Idealistic check is an indicator only, see documentation.
```

General use with MIP solvers

Read objectives & constraints from AMPL

- Store initially as linear coefficients + expression trees
- Analyze trees to determine if linearizable

Generate linearizations

- Walk trees to build linearizations (flatten)
- Define auxiliary variables (usually zero-one)
- Generate equivalent constraints

Solve

- Send to solver through its API
- Convert optimal solution back to the original AMPL variables
- Write solution to AMPL

Special Alternatives in *Gurobi*

Apply our linearization (`count`)

- Use Gurobi's linear API

Have Gurobi linearize (`or`, `abs`)

- Simplify and "flatten" the expression tree
- Use Gurobi's "general constraint" API
  - **`addGenConstrOr`** (resbinvar, [binvars])
    tells Gurobi: resbinvar = 1 iff at least one item in [binvars] = 1
  - **`addGenConstrAbs`** (resvar, argvar)
    tells Gurobi: resvar = |argvar|

Have Gurobi piecewise-linearize (`exp, sin`)

- Replace univariate nonlinear functions by p-l approximations
- Use Gurobi's "function constraint" API
  - **`addGenContstrExp`** (xvar, yvar)
    tells Gurobi: yvar = a piecewise-linear approximation of exp(xvar)

Special Alternatives in *Gurobi*

Apply our linearization (`count`)

- Use Gurobi's linear API

Have Gurobi linearize (`or`, `abs`)

- Simplify and "flatten" the expression tree
- Use Gurobi's "general constraint" API
  - **`addGenConstrOr`** (resbinvar, [binvars])
    tells Gurobi: resbinvar = 1 iff at least one item in [binvars] = 1
  - **`addGenConstrAbs`** (resvar, argvar)
    tells Gurobi: resvar = |argvar|

NEW

Have Gurobi apply its global optimizer (`exp`, `sin`)

- Replace univariate nonlinear functions by p-l approximations
- Use Gurobi's "function constraint" API
  - **`addGenContstrExp`** (xvar, yvar)
    tells Gurobi: yvar = exp(xvar)

Still Challenging

Convex functions

- Detection

Second-order cones

- Detection and transformation

Convex Functions

Test convexity

- Tree walk using rules for elementary functions
  - Linear functions are convex
  - Sum or maximum of convex functions is convex
  - Negative of a *concave* function is convex
  - A *nondecreasing* function of a convex function is convex
- Need rules for convex, concave, nondecreasing, nonincreasing

Test nonconvexity

- Check random line segments
- Check $\nabla^2 f(x)$ at random x values
- $\text{Min}_d \; g^T d + \frac{1}{2} d^T \nabla^2 f(x) d$ subject to $\|d\|^2 \leq \Delta$
  and stop when curvature is negative $(d^T \nabla^2 f(x) d < -\epsilon)$

Return "convex" or "nonconvex" or "inconclusive"

Examples

Searching line segments for nonconvexity

- John W. Chinneck, "Analyzing Mathematical Programs Using MProbe." *Annals of Operations Research* **104** (2001) 33-48. MProbe

"Disciplined" convex programming via convexity rules

- Michael Grant, Stephen Boyd, and Yinyu Ye, "Disciplined convex programming." In L. Liberti, N. Maculan, eds., *Global Optimization: From Theory to Implementation.* Nonconvex Optimization and Its Applications Series, Springer, Dordrecht, The Netherlands (2006) 155–210. CVX

Convexity rules + searching for negative curvature

- Robert Fourer, Chandrakant Maheshwari, Arnold Neumaier, Dominique Orban, Hermann Schichl, "Convexity and Concavity Detection in Computational Graphs: Tree Walks for Convexity Assessment." *INFORMS Journal on Computing* **22** (2010) 26-43.

Prospects for Implementation

Build into a local nonlinear solver

- Can report "globally optimal" when convex
- A lot of work to benefit only one solver

Build into a general solver interface

- Further complicates the interface library
- Benefits many solvers

Implement as a standalone system

- Run as a "pseudo-solver" that returns convexity status
- Use result as guidance for interpreting solver results

Still Challenging

Second-Order Cones

Basic forms

- $\sum_{i=1}^{n} x_i^2 \leq x_{n+1}^2, \quad x_{n+1} \geq 0$
- $\sum_{i=1}^{n} x_i^2 \leq x_{n+1}\, x_{n+2}, \quad x_{n+1} \geq 0, x_{n+2} \geq 0$

Detection

- Tree walk looks for SOC-equivalent formulations . . .
- SOC-representable functions in objectives or constraints
  - 2-norms, quadratic-linear ratios
  - Generalized geometric mean, p-norm
  - General affine function $a_i(\mathbf{f}_i\mathbf{x} + g_i)$ in place of x_i
- Additional objectives
  - Product of positive rational powers: $\prod_{i=1}^{n}(\mathbf{f}_i\mathbf{x} + g_i)^{\alpha_i}$
  - Logarithmic Chebychev: $\max_{i=1}^{n}|\log(\mathbf{f}_i\mathbf{x}) - \log(g_i)|$

Transformation

- Tree walk for each SOC equivalent that was found

Examples

Basic forms recognized by solvers

- Quadratic: many MIP solvers
- 2-norm: MOSEK

General Detection and Transformation

- Jared Erickson and Robert Fourer, "Detection and Transformation of Second-Order Cone Programming Problems in a General-Purpose Algebraic Modeling Language." Optimization Online, *https://optimization-online.org/2019/05/7194/*.

Second-Order Cones

Survey of Test Problems

13.5% of 1238 nonlinear problems were SOC-solvable

∗ from Vanderbei's CUTE & non-CUTE, and netlib/ampl

- 5.3% ordinary elliptic quadratic
- 1.7% basic SOC cases detected by solvers
- 6.5% additional SOC cases detected

A variety of forms detected

- `hs064` has $4/x_1 + 32/x_2 + 120/x_3 \leq 1$
- `hs036` minimizes $-x_1 x_2 x_3$
- `hs073` has $1.645\sqrt{0.28x_1^2 + 0.19x_2^2 + 20.5x_3^2 + 0.62x_4^2} \leq \ldots$
- `hs049` minimizes $(x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$
- `emfl_nonconvex` has $\sum_{k=1}^{2}\left(x_{jk} - a_{ik}\right)^2 \leq s_{ij}^2$

Prospects for Implementation

Extend recognition already built into solvers

- Quadratic and 2-norm cases only
  - more general forms are not passed to MIP solvers
  - functional forms are not passed to local nonlinear solvers
- Benefits all modeling languages
- Benefits only one solver

Build into a general solver interface

- Some SOC-equivalents are *very* complicated to process
  - consider implementing just the easier cases
- Benefits many solvers

Links

https://github.com/ampl/

- all AMPL open-source projects

https://github.com/ampl/mp

- *MP solver interface*

https://dev.ampl.com

- new AMPL development projects

https://colab.ampl.com/

- AMPL Colaboratory links