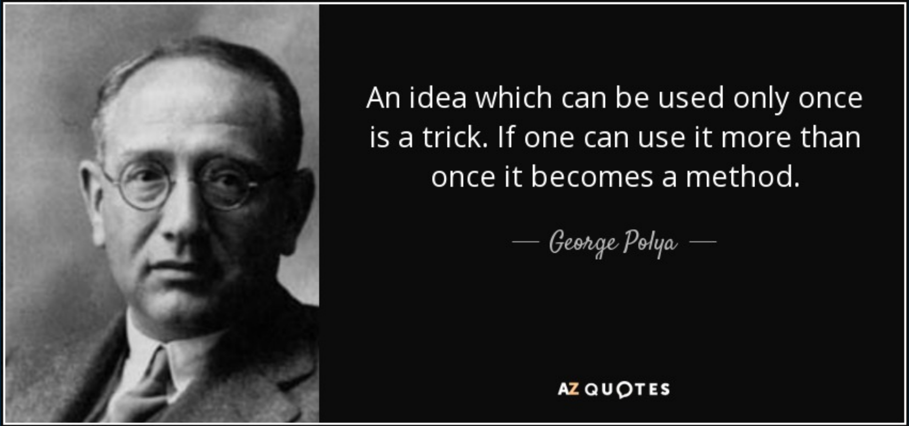
My Favorite Optimization Modeling ~~Tricks~~Methods

Rob Pratt

NC State University Operations Research Seminar

November 13, 2023

~~Tricks~~Methods



Outline

1. Binary Variables
2. Linearization
3. Decomposition
4. Network Reformulation
5. Sparsification
6. Strengthening Constraints
7. MILP Local Search

Binary Variables

Binary variable $x \in \{0, 1\}$ useful for modeling yes-no decisions

Constraint Type	Algebra
Conflict	$x + y \leq 1$
Implication	$x \leq y$
Partitioning (choose exactly one)	$\sum_{j \in J} x_j = 1$
Covering (choose at least one)	$\sum_{j \in J} x_j \geq 1$
Packing/Clique/SOS1 (choose at most one)	$\sum_{j \in J} x_j \leq 1$
Cardinality (choose exactly k)	$\sum_{j \in J} x_j = k$
Knapsack/Capacity	$\sum_{j \in J} a_j x_j \leq b$

Linearization

- Product of binary variables $\prod_{j \in J} x_j$
  - Replace product with binary variable z and impose linear constraints
  - $z \leq x_j$ for $j \in J$
  - $z \geq \sum_{j \in J} x_j - |J| + 1$
- Product of binary variables and bounded variable
- MIN, MAX, absolute value
- Ratio of linear constraints
- `solve linearize;` attempts linearization
- `expand / linearize;` expands linearized model
- `save mps|qps linearize;` saves linearized model to MPS or QPS format

Usual Linearization

$$x_{ij} \in \{0,1\}$$
$$y_{iji'j'} = x_{ij} \cdot x_{i'j'}$$

$$\rightsquigarrow$$

$$x_{ij} \in \{0,1\}$$
$$y_{iji'j'} \geq x_{ij} + x_{i'j'} - 1$$
$$y_{iji'j'} \leq x_{ij}$$
$$y_{iji'j'} \leq x_{i'j'}$$
$$y_{iji'j'} \geq 0$$

Compact Linearization

$$\sum_j x_{ij} = 1 \quad \text{for all } i$$

$$x_{ij} \in \{0, 1\}$$

$$y_{iji'j'} = x_{ij} \cdot x_{i'j'}$$

$$\rightsquigarrow$$

$$\sum_j x_{ij} = 1 \quad \text{for all } i$$

$$x_{ij} \in \{0, 1\}$$

$$\sum_j y_{iji'j'} = x_{i'j'}$$

$$0 \leq y_{iji'j'} \leq x_{ij}$$

Indicator Constraints

- Logical implication $y = 1 \implies \sum_j a_j x_j \leq b$
- `con C: y = 1 implies sum {j in JSET} a[j]*x[j] <= b;`
- Linearized via big-M constraint $\sum_j a_j x_j - b \leq M(1 - y)$
- Also supports $\geq$, $=$, and range constraints
- Consequent need not be linear but must be linearizable

Indicator Constraints Generalized

- Want to enforce $\sum_j a_j x_j \leq b \implies \sum_j c_j x_j \leq d$
- Split into two implications
  - $\sum_j a_j x_j \leq b \implies y = 1$
  - $y = 1 \implies \sum_j c_j x_j \leq d$
- Use contrapositive of the first ($P \implies Q$ is equivalent to $\neg Q \implies \neg P$)
  - `con C1:  y = 0 implies sum {j in JSET} a[j]*x[j] >= b + eps;`
  - `con C2:  y = 1 implies sum {j in JSET} c[j]*x[j] <= d;`
- `or.stackexchange.com/questions/10172`

Semicontinuous Variables

- Given constants $0 < \ell \leq u$
- Want to enforce $x \in \{0\} \cup [\ell, u]$
- Equivalently, $x = 0 \vee x \in [\ell, u]$
- Introduce binary variable y and use indicator constraints
  - `con C1:  y = 0 implies x = 0;`
  - `con C2:  y = 1 implies l <= x <= u;`
- Linearized via big-M constraints $\ell y \leq x \leq uy$

Disjoint Intervals

- Want to enforce $x \in [0,2] \cup [4,6] \cup [8,9]$
- Introduce binary variables y_1, y_2, y_3 and use indicator constraints

```
var x >= 0 <= 9;
var y {1..3} binary;
con Partition: y[1] + y[2] + y[3] = 1;
con C1: y[1] = 1 implies 0 <= x <= 2;
con C2: y[2] = 1 implies 4 <= x <= 6;
con C3: y[3] = 1 implies 8 <= x <= 9;
```

- Linearization via big-M constraints yields

$$
\begin{aligned}
x - 2 &\le (9-2)(1-y_1) \\
4 - x &\le (4-0)(1-y_2) \\
x - 6 &\le (9-6)(1-y_2) \\
8 - x &\le (8-0)(1-y_3)
\end{aligned}
$$

- Stronger linearization

$$0y_1 + 4y_2 + 8y_3 \le x \le 2y_1 + 6y_2 + 9y_3$$

Conjunctive Normal Form (CNF)

$$\bigwedge_{i \in I} \bigvee_{j \in J_i} z_{ij} \iff \left(\sum_{j \in J_i} z_{ij} \geq 1 \text{ for all } i \in I \right)$$

- R. Raman and I.E. Grossmann, "Relation Between MILP Modelling and Logical Inference for Chemical Process Synthesis," *Computers Chem. Engng.* **15** (1991), 73–84
- Three steps to convert proposition to CNF:
  1. Change $P \implies Q$ to $\neg P \vee Q$
  2. Push negation inward by De Morgan's laws
  3. Distribute $\vee$ over $\wedge$

Examples

Conflict

$$\neg(x \wedge y)$$

$$\neg x \vee \neg y$$

$$(1 - x) + (1 - y) \geq 1$$

$$x + y \leq 1$$

Implication

$$x \implies y$$

$$\neg x \vee y$$

$$(1 - x) + y \geq 1$$

$$x \leq y$$

Many others:

`or.stackexchange.com/search?q=%22conjunctive+normal+form%22`

No-Good Cuts

$$x \neq \hat{x}$$

$$\neg \left[\left(\bigwedge_{j:\hat{x}_j=1} x_j \right) \wedge \left(\bigwedge_{j:\hat{x}_j=0} \neg x_j \right) \right]$$

$$\neg \left(\bigwedge_{j:\hat{x}_j=1} x_j \right) \vee \neg \left(\bigwedge_{j:\hat{x}_j=0} \neg x_j \right)$$

$$\left(\bigvee_{j:\hat{x}_j=1} \neg x_j \right) \vee \left(\bigvee_{j:\hat{x}_j=0} x_j \right)$$

$$\sum_{j:\hat{x}_j=1} (1 - x_j) + \sum_{j:\hat{x}_j=0} x_j \geq 1$$

Decomposition

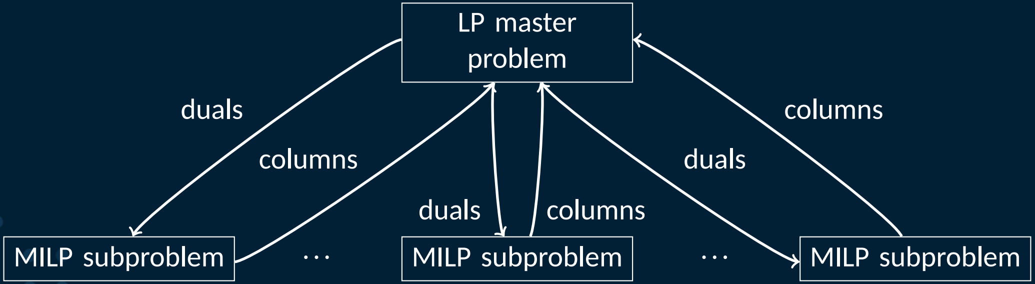
- Completely independent problems: use COFOR or groupBy
- Loosely coupled problems: exploit block-angular structure in constraint matrix
  - Dantzig-Wolfe decomposition
  - Benders decomposition

Dantzig-Wolfe Versus Benders

Dantzig-Wolfe Decomposition	Benders Decomposition
$\begin{pmatrix} A_1 & A_2 & \cdots & A_{\lvert K\rvert} \\ B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_{\lvert K\rvert} \end{pmatrix}$	$\begin{pmatrix} A_1 & B_1 & & & \\ A_2 & & B_2 & & \\ \vdots & & & \ddots & \\ A_{\lvert K\rvert} & & & & B_{\lvert K\rvert} \end{pmatrix}$
Complicating/linking constraints	Complicating/linking variables
LP master problem	MILP master problem
MILP subproblems	LP subproblems
Column generation	Row generation
Requires further branching	

Dantzig-Wolfe Decomposition

- If complicating constraints are omitted, resulting problem is easy
- LP master problem combines columns and finds optimal dual variables
- MILP subproblems generate negative reduced cost columns from dual variables
- Iterate between master and subproblems until optimality gap is small enough
- Requires further branching

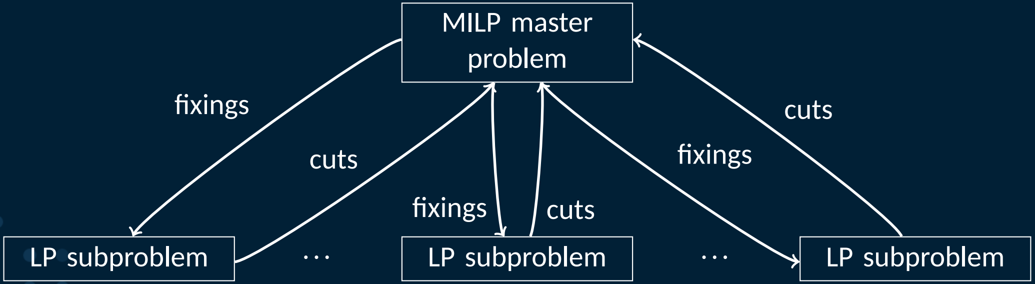


Decomposition Algorithm

- Accessible in OPTMODEL, OPTLP, OPTMILP procedures
- User conveys block structure via `.block` constraint suffix
- `solve with LP|MILP / decomp=(method=user);`
- Master and subproblems generated and solved automatically and in parallel
- For some block-angular problems, dramatic performance improvements over branch-and-cut algorithm
- Automatically detects identical subproblems and uses Ryan-Foster branching if applicable

Benders Decomposition

- If complicating variables are fixed, resulting problem is easy
- Benders (1962): take integer variables as complicating
- MILP master problem recommends values for complicating variables
- LP subproblems generate optimality and feasibility cuts from dual variables
- Iterate between master and subproblems until optimality gap is small enough



Combinatorial Benders Decomposition

- Classical: MILP master, LP subproblems
- Combinatorial: MILP master, arbitrary subproblems
- If master variables are binary…
  - Benders feasibility cuts are *no-good* constraints that enforce $x \neq \hat{x}$:

$$\sum_{j:\hat{x}_j=0} x_j + \sum_{j:\hat{x}_j=1} (1 - x_j) \geq 1$$

  - Benders optimality cuts are *big-M* constraints that enforce $x = \hat{x} \implies \eta \geq \hat{\eta}$

Implementations

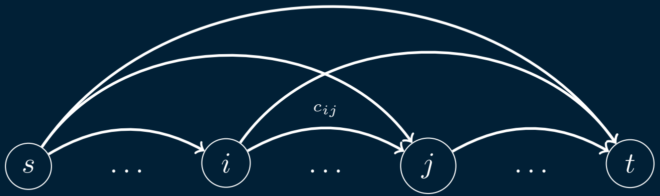
- Poor man's approach: solve MILP in each outer iteration
- Better: one tree, generate Benders cut for each new integer feasible solution
- Competitors
  - SCIP: `https://www.scipopt.org/doc/html/BENDDECF.php`
  - CPLEX: `https://www.ibm.com/docs/en/icos/20.1.0?topic=optimization-benders-algorithm`
- Plans for SAS release in early 2024

Network Reformulation

- Multiperiod problem with binary variable y_p in each period $p \in \{1, \ldots, n\}$
- Node set $\{0, \ldots, n+1\}$, with source $s = 0$ and sink $t = n+1$
- For $i < j$, introduce binary arc variable

$$x_{ij} = \begin{cases} 1 & \text{if } y_i = 1 \text{ and } y_j = 1 \text{ but } y_p = 0 \text{ for } i < p < j \\ 0 & \text{otherwise} \end{cases}$$

- Find shortest path from source to sink



Network Reformulation

- Possibly multiple networks with shared resources
- Integer network flow problem with few side constraints
- Often solves at root node of branch-and-cut tree
- Many more variables: $O(n^2)$ instead of $O(n)$
- Difficult side constraints in initial formulation correspond to removal of arcs
  - Upper limit on number of consecutive cashout hours
  - Lower limit on number of hours between replenishments

Sparsification

- Sometimes better to make problem bigger but sparser
- Introduce explicit variable and constraint for repeated expression

```
var NewVar {ISET};
con NewCon {i in ISET}:
   NewVar[i] = sum {j in JSET} a[i,j]*X[j];
```

- Least squares

```
/* original */
min SSE = sum {i in OBS} (sum {j in FEATURES} x[i,j]*Beta[j] - y[i])^2;

/* reformulation */
var Error {OBS};
con ErrorCon {i in OBS}:
   Error[i] = sum {j in FEATURES} x[i,j]*Beta[j] - y[i];
min SSE = sum {i in OBS} Error[i]^2;
```

Strengthening Conflict Constraints to Clique Constraints

- Binary variables x_i, x_j, x_k
- Original constraints

$$x_i + x_j \leq 1, \quad x_i + x_k \leq 1, \quad x_j + x_k \leq 1$$

- Replace with

$$x_i + x_j + x_k \leq 1$$

- Cuts off fractional solution $x = (1/2, 1/2, 1/2)$

```
set <num,num> ID_NODE;
solve with network / clique=(maxcliques=ALL) links=(include=CONFLICTS) out=(cliques=ID_NODE);
set CLIQUES init {};
set NODES_c {CLIQUES} init {};
for {<c,i> in ID_NODE} do;
   CLIQUES     = CLIQUES     union {c};
   NODES_c[c] = NODES_c[c] union {i};
end;
con Clique {c in CLIQUES}:
   sum {i in NODES_c[c]} X[i] <= 1;
```

Strengthening in the Presence of Clique Constraints

- Binary variables x_i, arbitrary variables y_j
- Original constraints

$$x_i + \sum_j a_j y_j \leq b \qquad \text{for all } i \qquad (1)$$

$$\sum_i x_i \leq 1 \qquad (2)$$

- Replace (1) with (3)

$$\sum_i x_i + \sum_j a_j y_j \leq b \qquad (3)$$

- `or.stackexchange.com/questions/6187`

Strengthening in the Presence of Variable Upper Bounds

- Nonnegative variables x_i, binary variable y
- Via explicit constraints or probing, suppose $y = 0 \implies x_i = 0$ for all i
  - $x_i \leq M_i y$ for all i
  - $\sum_i a_i x_i \leq by$
- Original constraints

$$y = 0 \implies x_i = 0 \qquad \text{for all } i \tag{4}$$

$$\sum_i c_i x_i \leq d \tag{5}$$

- Replace (5) with (6)

$$\sum_i c_i x_i \leq dy \tag{6}$$

MILP Local Search

- Very Large Neighborhood Search, Ruin and Recreate, Solution Polishing
- Improvement heuristic
- Fix subset of variables and solve resulting MILP, much easier than original MILP
- Current solution always integer feasible, so use PRIMALIN
- Repeat as many times as you like, fixing random(?) subset of variables
- Terminate each MILP early to avoid spending too much time on any one subinstance

Additional Links

- Compact Linearization
  - ATM Cash Management example in DECOMP documentation
- Benders Decomposition
  - Linear Optimization in SAS/OR Software: Migrating to the OPTMODEL Procedure
- Network Reformulation
  - The Traveling Baseball Fan Problem
  - Monitor Assignment for Students with Disabilities Using SAS Optimization
- Strengthening Constraints
  - Using SAS/OR to Optimize the Layout of Wind Farm Turbines
  - Why Venue Optimization is Critical and How It Works
  - Machine Learning for Combinatorial Optimization competition 2021: video, paper

support.sas.com