Specialized Strategies for Products of Binary Variables

GUROBI OPTIMIZATION

The World's Fastest Solver

Ed Klotz, Ph.D. (klotz@gurobi.com)
February 2021

Products of binaries

- **Problem formulation:**

$$\min [\, c^T x \,] + x^T Q x \qquad // \; no\ assumptions\ on\ convexity\ of\ x^T Q x$$
$$[\; s.t. \quad Ax \sim b \;]$$
$$x \in \{0,1\}$$

- Possibly nonconvex MIQP
- Can reformulate constraints into objective using penalties (QUBO)
  - Good formulation for Quantum Annealers, not so good for solvers like Gurobi
  - https://www.springerprofessional.de/en/quantum-bridge-analytics-i-a-tutorial-on-formulating-and-using-q/17436666

Outline

- Products of binaries fundamentals
- Solver options and parameters
- Working with the existing formulation
- Reformulations

Products of binaries fundamentals

- **Solution strategies**

  - Solve as convex or nonconvex MIQP
    - Still must deal with the quadratic objective
    - Starting with version 9.0, Gurobi can solve nonconvex MIQPs (and MIQCPs)

  - Transform into a convex MIQP or a MILP
    - Convexification of objective ($x_i^2 = x_i$ for binary variables)
      - $x_i x_j = x_i x_j + \underbrace{(x_i^2 - x_i)}_{0} + \underbrace{(x_j^2 - x_j)}_{0} = \underbrace{(x_i^2 + x_i x_j + x_j^2)}_{\text{convex}} - x_i - x_j$
      - $x^T Q x = x^T Q x + \underbrace{x^T D x - d^T x}_{0} = x^T \underbrace{(Q + D)}_{\text{PSD}} x - d^T x$

Products of binaries fundamentals

- PreQLinearize = 0: Convexification of objective ($x_i^2 = x_i$ for binary variables)
- A nonconvex MIQP becomes a convex one without adding constraints
  - But there is no free lunch
    - $x_i x_j = x_i x_j + \underbrace{(x_i^2 - x_i)}_{0} + \underbrace{(x_j^2 - x_j)}_{0} = \underbrace{(x_i^2 + x_i x_j + x_j^2)}_{\text{convex}} \underbrace{- x_i - x_j}_{\text{relaxation}}$
    - Consider $x_i = x_j = .5$ in

$$\min x_i x_j \ \ s.t$$
$$x_i + x_j = 1$$
$$x_i, x_j \geq 0$$

  - Objective value in relaxation is -.25
  - Potential for negative dual bound values for convexified model that has an obvious lower bound of 0 in the original model
  - As magnitude of D increases, so does the weakness in the dual bound of the convexified problem

Products of binaries fundamentals

- **Solution strategies**
  - Linearize a convex or nonconvex MIQP into a MILP
    - Simplest linearization technique: do the following for each product of binaries in the model (PreQLinearize=1)
      - $z_{ij} = x_i x_j$

        $\left.\begin{array}{l} z_{ij} \leq x_i \\ z_{ij} \leq x_j \end{array}\right\}$ (only need these two if objective pushes z_{ij} up)

        $z_{ij} \geq x_i + x_j - 1 \}$ (only need this one if objective pushes z_{ij} down)
      - Add the 3 linear constraints to the model
      - Replace each occurrence of $x_i x_j$ in the model with z_{ij}
    - We've transformed a (possibly nonconvex MIQP) into a MILP
      - Benefit from various Gurobi features available for MILP but not MIQP
      - Still no free lunch
        - We added 1-3 constraints for each product of binaries.

Products of binaries fundamentals

- **Solution strategies**
  - Linearize a convex or nonconvex MIQP into a MILP
    - A less straightforward but more compact linearization technique
      - Consider the MIPLIB 2010 model neos-911970

```
Minimize
  C0001 + C0002 + C0003 + C0004 + C0005 + C0006 + C0007 + C0008 + C0009
   + C0010 + C0011 + C0012 + C0013 + C0014 + C0015 + C0016 + C0017 + C0018
   + C0019 + C0020 + C0021 + C0022 + C0023 + C0024 + C0025 + C0026 + C0027
   + C0028 + C0029 + C0030 + C0031 + C0032 + C0033 + C0034 + C0035 + C0036
   + C0037 + C0038 + C0039 + C0040 + C0041 + C0042 + C0043 + C0044 + C0045
   + C0046 + C0047 + C0048
Subject To
R0001: - C0025 + 5.43 C0049 + 5.56 C0073 + 5.2 C0097 + 5.4 C0121 + 5 C0145
   + 4.39 C0169 + 4.07 C0193 + 4.56 C0217 + 4.03 C0241 + 3.3 C0265
   + 4.39 C0289 + 5.64 C0313 + 5.9 C0337 + 3.57 C0361 + 6.4 C0385
   + 3.94 C0409 + 4.5 C0433 + 4.67 C0457 + 3.88 C0481 + 4.18 C0505
   + 4.31 C0529 + 4.63 C0553 + 4.74 C0577 + 5.5 C0601 + 5.1 C0625
   + 5.1 C0649 + 4.2 C0673 + 6.5 C0697 + 5.95 C0721 + 5.88 C0745
   + 5.77 C0769 + 5.36 C0793 + 5.64 C0817 + 5.04 C0841 + 5.53 C0865 <= 6.5
…
```

Penalty variable on soft knapsack constraint R0001

All other variables in constraint are binary

Products of binaries fundamentals

- **A less straightforward but more compact linearization technique**
  - Look at a simpler version of this soft knapsack constraint

Minimize
C0001 + C0002 + C0003 + C0004 + C0005 + C0006 + C0007 + C0008 + C0009
+ C0010 + C0011 + C0012 + C0013 + C0014 + C0015 + C0016 + C0017 + C0018
+ C0019 + C0020 + C0021 + C0022 + C0023 + C0024 + C0025 + C0026 + C0027
+ C0028 + C0029 + C0030 + C0031 + C0032 + C0033 + C0034 + C0035 + C0036
+ C0037 + C0038 + C0039 + C0040 + C0041 + C0042 + C0043 + C0044 + C0045
+ C0046 + C0047 + C0048
Subject To
R0001a: - C0025 + 4.2 C0673 + 6.5 C0697 + 5.95 C0721 <= 6.5

Knapsack capacity matches largest knapsack weight

C0673 = C0697 = 1 --> C0025 = 4.2 + 6.5 - 6.5 = 4.2
C0673 = C0721 = 1 --> C0025 = 4.2 + 5.95 - 6.5 = 3.65
C0697 = C0721 = 1 --> C0025 = 6.5 + 5.95 - 6.5 = 5.95
C0673 = C0697 = C0721 = 1 -> C0025 = 4.2 + 5.95 = 10.15

R0001a provides a linear representation of
C0025 = 4.2 C0673*C0697 + 3.65 C0673*C0721 + 5.95 C0697*C0721 - 3.65 C0697*C0673*C0721

Products of binaries fundamentals

- **A less straightforward but more compact linearization technique**
  - Implications for the full constraint R0001

R0001: - C0025 + 5.43 C0049 + 5.56 C0073 + 5.2 C0097 + 5.4 C0121 + 5 C0145
+ 4.39 C0169 + 4.07 C0193 + 4.56 C0217 + 4.03 C0241 + 3.3 C0265
+ 4.39 C0289 + 5.64 C0313 + 5.9 C0337 + 3.57 C0361 + 6.4 C0385
+ 3.94 C0409 + 4.5 C0433 + 4.67 C0457 + 3.88 C0481 + 4.18 C0505
+ 4.31 C0529 + 4.63 C0553 + 4.74 C0577 + 5.5 C0601 + 5.1 C0625
+ 5.1 C0649 + 4.2 C0673 + 6.5 C0697 + 5.95 C0721 + 5.88 C0745
+ 5.77 C0769 + 5.36 C0793 + 5.64 C0817 + 5.04 C0841 + 5.53 C0865 <= 6.5

C00025 = (linear combinations of all the bilinear terms) – (linear combinations of larger multilinear terms)

  - Could represent this complicated multilinear expression via a single constraint
  - Suspect the creator of this model was just thinking about soft knapsack constraints (no info on MIPLIB set regarding model origins).
  - Can we modify this to help us linearize an expression just involving bilinear terms?

Products of binaries fundamentals

- **A less straightforward but more compact linearization technique**
  - Consider a different version of this constraint:

R0001': - C0025 + 5.43 C0049 + 5.56 C0073 + 5.2 C0097 + 5.4 C0121 + 5 C0145
+ 4.39 C0169 + 4.07 C0193 + 4.56 C0217 + 4.03 C0241 + 3.3 C0265
+ 4.39 C0289 + 5.64 C0313 + 5.9 C0337 + 3.57 C0361 + 6.4 C0385
+ 3.94 C0409 + 4.5 C0433 + 4.67 C0457 + 3.88 C0481 + 4.18 C0505
+ 4.31 C0529 + 4.63 C0553 + 4.74 C0577 + 5.5 C0601 + 5.1 C0625
+ 5.1 C0649 + 4.2 C0673 + **166.76 C0697** + 5.95 C0721 + 5.88 C0745
+ 5.77 C0769 + 5.36 C0793 + 5.64 C0817 + 5.04 C0841 + 5.53 C0865 <= **166.76**

166.76 = sum of all knapsack weights except for C0697

- All sums of knapsack weights other than C0697 will be <= the rhs
  - → all multilinear expressions not involving C0697 contribute 0 violation to this soft constraint
- If C0697 = 1 and any other binary variable = 1 we get a contribution of the other binary variable's coefficient to the violation (e.g C0049 = 1 contributes 5.43 of violation).
- C0025= 5.43 C0049*C0697 + 5.56 C0073*C0697 + ... + 5.534 C0865*C0697
  - C0025 represents precisely a quadratic expression involving C0697 and other binaries

Products of binaries fundamentals

- A less straightforward but more compact linearization technique
  - C0025= 5.43 C0049*C0697 + 5.56 C0073*C0697 + ... + 5.534 C0865*C0697
    - C0025 represents precisely a quadratic expression involving C0697 and other binaries
  - Gurobi's PreQLinearize = 2 setting uses this to do a more compact linearization
  - $q_1 y * x_1 + \cdots + q_n y * x_n \ \ (y, x_j \ binary)$ is linearized as

    $q_1 x_1 + \cdots + q_n x_n + qy \ - p \leq q \qquad (q = \sum_{j=1}^{n} q_j) \qquad // \ q_j > 0.$

    $p = q_1 y * x_1 + \cdots + q_n y * x_n$

Products of binaries fundamentals

- **Summary of Gurobi PreQLinearize settings: Still No Free Lunch**
  - PreQLinearize = 0: Convexify the nonconvex quadratic objective
    - Move from nonconvex MIQP to convex MIQP👍
    - No additional constraints👍
    - Miss out on MILP features absent from convex MIQP solver👎
    - Counterintuitive dual bound values that suggest possibly weak relaxations👎
  - PreQLinearize = 1: Linearize the nonconvex quadratic objective with new variable and constraints for each bilinear objective term
    - Move from nonconvex MIQP to MILP👍💪
      - Fairly strong MILP formulation
    - Each bilinear term in the quadratic objective introduces one new variable and one or two additional linear constraints👎
  - PreQLinearize = 2: Linearize using the soft knapsack constraints
    - Move from nonconvex MIQP to MILP👍
    - Multiple bilinear terms modelled with one additional variable and constraint👍
    - Weaker MILP formulation 👎

Products of binaries fundamentals

- **Gurobi PreQLinearize settings**
  - Gurobi's default logic to choose works fairly well.
  - Use node log to determine the default selection
    - Compare original model with presolved model

Optimize a model with 1 rows, 50 columns and 50 nonzeros
Model fingerprint: 0xe647a136
Model has 1225 quadratic objective terms
Variable types: 0 continuous, 50 integer (50 binary)
Coefficient statistics:
  Matrix range   [1e+00, 1e+00]
  Objective range [0e+00, 0e+00]
  QObjective range [2e-01, 2e+01]
  Bounds range   [1e+00, 1e+00]
  RHS range   [1e+01, 1e+01]
Found heuristic solution: objective 246.2637697
Presolve time: 0.00s
Presolved: 1 rows, 50 columns, 50 nonzeros
Presolved model has 1275 quadratic objective terms
Variable types: 0 continuous, 50 integer (50 binary)

Default setting; convexification (PreQLinearize = 0) selected.
No additional constraints; number of quadratic terms has increased due to convexification (one new nonzero the 0 diagonal term associated the square of each of the 50 variables)

Products of binaries fundamentals

- **Gurobi PreQLinearize settings**
  - Gurobi's default logic to choose works fairly well.
  - Use node log to determine the default selection
    - Compare original model with presolved model

Optimize a model with 1 rows, 50 columns and 50 nonzeros
Model fingerprint: 0xe647a136
Model has 1225 quadratic objective terms
Variable types: 0 continuous, 50 integer (50 binary)
Coefficient statistics:
  Matrix range   [1e+00, 1e+00]
  Objective range [0e+00, 0e+00]
  QObjective range [2e-01, 2e+01]
  Bounds range   [1e+00, 1e+00]
  RHS range   [1e+01, 1e+01]
Found heuristic solution: objective 246.2637697
Presolve time: 0.00s
Presolved: 1226 rows, 1275 columns, 3725 nonzeros
Variable types: 0 continuous, 1275 integer (1275 binary)

Set PreQLinearize = 1 (simple linearization).
One additional constraint and variable for each bilinear term; no quadratic objective terms in presolved model.

Products of binaries fundamentals

- **Gurobi PreQLinearize settings**
  - Gurobi's default logic to choose works fairly well.
  - Use node log to determine the default selection
    - Compare original model with presolved model

Optimize a model with 1 rows, 50 columns and 50 nonzeros
Model fingerprint: 0xe647a136
Model has 1225 quadratic objective terms
Variable types: 0 continuous, 50 integer (50 binary)
Coefficient statistics:
  Matrix range   [1e+00, 1e+00]
  Objective range [0e+00, 0e+00]
  QObjective range [2e-01, 2e+01]
  Bounds range   [1e+00, 1e+00]
  RHS range   [1e+01, 1e+01]
Found heuristic solution: objective 246.26
Presolve time: 0.00s
Presolved: 50 rows, 99 columns, 1373 nonzeros
Variable types: 48 continuous, 51 integer (51 binary)

Set PreQLinearize = 2 (compact linearization).
One additional constraint (49 total) for all bilinear terms involving a single variable; 49 additional penalty variables no quadratic objective terms in presolved model.

An Interesting Example

- **The p-Dispersion-Sum problem**
  - Given a set of n points with distances dij between points i and j, find the subset of k points that maximizes the sum of the distances

$$
\begin{aligned}
Max\ & \textstyle\sum_{i<j} d_{ij} x_i x_j \\
s.t.\ & \textstyle\sum_{j=1}^{n} x_j = k \\
& x_j \in \{0,1\}
\end{aligned}
$$

- **Example discussed in Practical Guidelines for Solving Difficult MILPs (https://www.sciencedirect.com/science/article/abs/pii/S1876735413000020**
- **Broader discussion in http://yetanothermathprogrammingconsultant.blogspot.com/2019/06/maximum-dispersion.html**
- **We'll come back to this later.**

Solver options and parameters

- **Solve directly as (non)convex MIQP**
  - Via Spatial Branch and Bound if using Gurobi
  - Set NonConvex parameter to 2
  - Not the best choice on the models considered in this presentation.
- **Transform into a MILP or convex MIQP**
  - Choose between convexification and linearization (PreQlinearize parameter)
  - Convexification
    - More compact formulation, but weaker
  - Linearization
    - Larger, stronger formulation
    - More opportunities to provide polyhedral cuts, use other MILP features
    - Can increase intensity of RLT and BQP cuts
      - For general bilinear terms, but particularly effective on products of binaries
      - More later
    - Zero-Half cuts when PreQLinearize = 1
      - Good in general on constraints with all binaries, +-1, +-2 coefficients

Results for the p-Dispersion-Sum problem, n=50, k=10

- $$\begin{aligned} &\mathbf{Max} \textstyle\sum_{i<j} d_{ij} x_i x_j \\ s.t. \quad &\textstyle\sum_{j=1}^{n} x_j = k \\ &\quad x_j \in \{0, 1\} \end{aligned}$$
- Gurobi 9.1, 2 Intel(R) Xeon(R) CPU E3-1240 v5 @ 3.50GHz quad core processors
- Defaults: Out of memory with gap of 52.3% after 1.35 hours
  - Gurobi by default chose to convexify rather than linearize
- PreQLinearize = 1: Optimal, 3.62 hours
- PreQLinearize = 2, RLTCuts = 2: Optimal, 1.55 hours
- Can we do better by tightening the formulation?

Working with the existing formulation

- LP relaxation feasible region is the convex hull of the integer feasible points
  - Won't be able to use LP-based polyhedral cuts on this direct formulation
  - Previous results indicate just linearizing (PreQLinearizing=1) is better, but still not particularly effective given the problem size
- http://yetanothermathprogrammingconsultant.blogspot.com/2019/06/maximum-dispersion.html describes multiple ways to derive a single cut that uses both original x binary variables and the linearization variables z
- $$\begin{aligned} Max\ & \textstyle\sum_{i<j} d_{ij} x_i x_j \\ s.t.\ & \textstyle\sum_{j=1}^{n} x_j = k \\ & x_j \in \{0,1\} \end{aligned}$$

  $\Rightarrow$

  $$\begin{aligned} Max\ & \textstyle\sum_{i<j} d_{ij} z_{ij} \\ s.t.\ & \textstyle\sum_{j=1}^{n} x_j = k \end{aligned}$$

  <linearization constraints>

  $$\boxed{\textstyle\sum_{i<j} z_{ij} = k * (k-1)/2}$$

  $$x_j \in \{0,1\}$$
- Run time with cut drops from hours to < a minute
- Blog describes multiple ways to derive this cut, but how do we do it generically in a way that extends to other models?

Working with the existing formulation

GUROBI OPTIMIZATION

- $\boldsymbol{Max} \sum_{i<j} d_{ij} x_i x_j$

  $s.t. \quad \sum_{j=1}^{n} x_j = k$

  $x_j \in \{0,1\}$

$\Rightarrow$

$\boldsymbol{Max} \sum_{i<j} d_{ij} z_{ij}$

$s.t. \quad \boxed{\sum_{j=1}^{n} x_j} = k$

<linearization constraints>

$\boxed{\sum_{i<j} z_{ij} = k*(k-1)/2}$

$x_j \in \{0,1\}$

- Generic approach #1: RLT and aggregate (from the blog):

$\boxed{x_i * (\sum_{j=1}^{n} x_j) = k * x_i}$ $\Rightarrow$ $x_i * (\sum_{j<i} x_j + \sum_{j>i} x_j) + x_i^2 = k * x_i$

$x_i^2 = x_i$

$\Rightarrow$ $\sum_{j<i} z_{ij} + \sum_{j>i} z_{ij} = (k-1) * x_i$

(add all n such constraints: $\sum_{i=1}^{n} (\sum_{j<i} z_{ij} + \sum_{j>i} z_{ij}) = (k-1) * \boxed{\sum_{i=1}^{n} x_i}$

$\Rightarrow \sum_{j \neq i} z_{ij} = (k-1) * k \Rightarrow 2 * \sum_{i<j} z_{ij} = k*(k-1) \Rightarrow \sum_{i<j} z_{ij} = k*(k-1)/2$

Working with the existing formulation

- RLT approach of last slide
  A bit complex, but generic
  and effective

$$Max \sum_{i<j} d_{ij} z_{ij}$$
$$s.t. \quad \sum_{j=1}^{n} x_j = k$$
<linearization constraints>
$$\boxed{\sum_{i<j} z_{ij} = k * (k-1)/2}$$
$$x_j \in \{0, 1\}$$

- Generic approach #2: just use the "Padberg Graph"
  - Padberg, The Boolean Quadric Polytope: Some Characteristics, Facets and Relatives

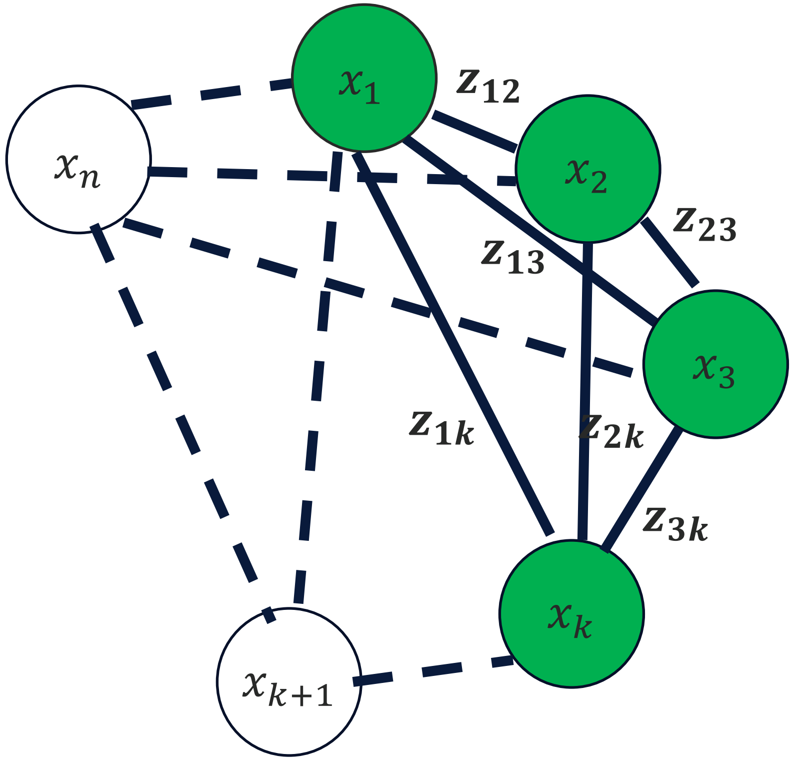
$$z_{ij} = x_i x_j$$
$$z_{ij} \le x_i$$
$$z_{ij} \le x_j$$
$$z_{ij} \ge x_i + x_j - 1$$

- Either way, we are determining the number of z variables that must be 1

Working with the existing formulation

- **Padberg graph for our dispersion problem**

Complete graph since $d_{ij} > 0$



$$\boldsymbol{Max\ \sum_{i<j} d_{ij} z_{ij}}$$

$$\boldsymbol{s.t.\quad \sum_{j=1}^{n} x_j = k}$$

<linearization constraints>

$$\boxed{\boldsymbol{\sum_{i<j} z_{ij} = k * (k-1)/2}}$$

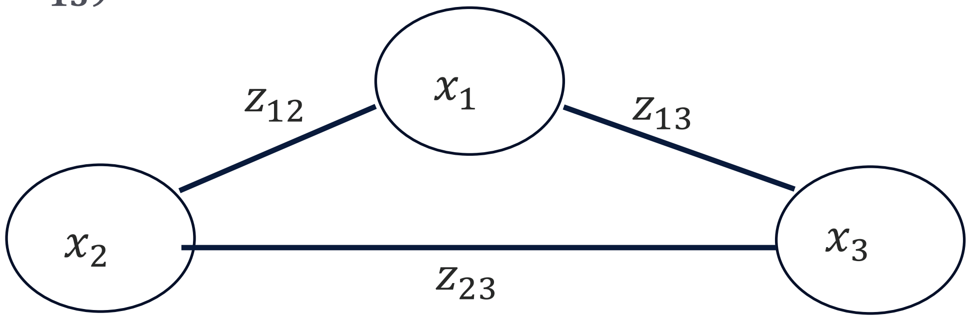
$$\boldsymbol{x_j \in \{0,1\}}$$

- **Given that k of the x variables must be 1, how many of the z variables must be 1?**
  - WLOG, set the first k x variables to 1
  - Induces a complete subgraph on the green nodes associated with $x_1, \ldots, x_k$
  - Each edge in the subgraph identifies a z variable that must be 1
  - There are $\boldsymbol{k * (k-1)/2}$ such edges

Source: http://orwe-conference.mines.edu/files/IOS2018SpatialPerfTuning.pdf

Working with the existing formulation

- **Padberg, The Boolean Quadric Polytope: Some Characteristics, Facets and Relatives**
  - He used the Padberg graph in a generic manner that just used the quadratic objective
  - Generate cuts even when the problem has no constraints
    - Example: $\boldsymbol{x_1 + x_2 + x_3 - (z_{12} + z_{23} + z_{13}) \leq 1}$
    - Can prove by contradiction
    - Or by induction
      - Extends to cliques of larger size
    - Or by deriving as a zero half cut
      - But Padberg figured it out first
    - Or via facet defining inequalities



  - **Gurobi's BQP cut feature makes use of this with cliques of size 3**
    - **Traction for cut generation when model has few or no constraints**
  - Adding these Padberg Cuts for cliques of size 4 or more may be useful
    - For 9.1.x with x >= 1, can set GURO_PAR_MOREBQPCUTS to 1
    - For next major release and beyond, more refined improvements will be integrated with no need for setting a hidden parameter.

Working with the existing formulation

- $Max \sum_{i<j} d_{ij} x_i x_j$

  $s.t. \quad \sum_{j=1}^{n} x_j = k$

  $x_j \in \{0, 1\}$

$Max \sum_{i<j} d_{ij} z_{ij}$

$s.t. \quad \boxed{\sum_{j=1}^{n} x_j} = k$

<linearization constraints>

$\boxed{\sum_{i<j} z_{ij} = k * (k-1)/2}$

$x_j \in \{0, 1\}$

- Best time on original model: 1.55 hours (PreQLinearize=2, RLTCuts=2)
- Add cardinality cut to Gurobi presolved model with PreQLinearize=1: 18 seconds
- Do the 3 constraint linearization described in this presentation: 8 seconds
- Add the cardinality cut to 3 constraint linearization model: 5 seconds

Working with the existing formulation

- **Does the success tightening the n=50, k=10 p-dispersion-sum model carry over to other models?**
- **Consider the publicly available QPLIB models on which Gurobi exceeds or comes close to the one hour time limit on the Mittelmann benchmark.**
- Model 3772
  - Convexify objective (PreQLinearize=0): 11.5% gap after 2 hours
  - Linearize objective (PreQLinearize=1, Gurobi's default choice): Optimal, 27.88 minutes
  - Best non default settings: ZeroHalfCuts=2, RLTCuts=2: Optimal, 14.22 minutes
  - No success so far tightening the formulation
    - All original binaries can get set to 0
      - All linearization variables can be set to 0
    - No cardinality type cut like in the p-dispersion sum model is available

Working with the existing formulation

- **More QPLIB models**
- **Model 3775**
  - Convexify objective (PreQLinearize=0, Gurobi's default choice): 27.32 minutes
  - Linearize objective (PreQLinearize=1) 30.73 minutes
  - Best non default settings found: 22.67 minutes (PreQLinearize=1, RLTCuts=2)
  - Tightening the existing formulation
    - No explicit cardinality constraints on the original variables like in the p-dispersion-sum problem
    - Q matrix is not dense like in the p-dispersion-sum problem
    - But we can ask a similar question: what is the minimum number of linearization binaries that must be set to 1 in a feasible solution for the linearized MIQP?
      - Formulate and solve the appropriate subMIP

How many z variables must be 1?

- **Model 3775 (continued)**
  - SubMIP solve for case where $Q_{ij} \geq 0$:

$$\min c^T x + x^T Qx \quad [\, s.t. \;\; Ax \sim b \,] \quad x \in \{0,1\} \xrightarrow{\text{linearize}} \min c^T x + \boxed{d^T z} \quad [\, s.t. \;\; Ax \sim b \,] \quad \boxed{E_1 x + E_2 z \leq p} \quad x, z \in \{0,1\}$$

$$\xrightarrow{\text{subMIP}} \quad e=(1,1,\ldots,1) \quad \min e^T z \quad [\, s.t. \;\; Ax \sim b \,] \quad E_1 x + E_2 z \leq p \quad x, z \in \{0,1\}$$

  - Cardinality cut: $\boldsymbol{e^T z \geq z^*}$
  - Best time on original model: 22.67 minutes (PreQLinearize=1, RLTCuts=2)
  - Add cardinality cut to Gurobi presolved model with PreQLinearize=1: 15.40 minutes*
  - Do the 3 constraint linearization described in this presentation: 49.18 minutes
  - Add the cardinality cut to 3 constraint linearization model: 8.95 minutes*

*: Includes 32 sec. for subMIP solve

Working with the existing formulation

- More QPLIB models
- Model 3587
  - Convexify objective (PreQLinearize=0): Hopeless
  - Linearize objective (PreQLinearize=1, Gurobi's default choice) : 62.63 minutes
  - Best non default settings found: 49.55 minutes (RLTCuts = ZeroHalfCuts = 2)
  - subMIP yields a useful cardinality cut
    - Gurobi linearized model: 15.37 minutes
    - 3 constraint linearized model 12.58 minutes
    - Includes 0.5 sec. for sub MIP solve
    - Similar results giving 3 constraint linearized model to Gurobi without the cut
- Model 3614
  - Appears to be a different instance of the same model as 3587
  - Over 3x speedup adding cardinality cut to 3 constraint linearized model compared to defaults (subMIP solve time only 0.02 sec.)

Working with the existing formulation

- **QPLIB models summary**
  - Discarded model 0752 due to high performance variability

Model name	Defaults (minutes)	Best non default	Cardinality Cut
3772	27.88	14.22	---------
3775	27.32	22.67	8.95
3587	62.63	49.55	12.58
3614	11.65	9.83	3.07

- Found non default settings to improve performance on all 4 models
- Cardinality cut offered the best performance on the 3 of 4 models
  - And all 3 on which it could be created.

Working with the existing formulation

- **Other results with sub MIP cuts**
- **Quadratic Assignment Problems (QAPs) from the QAPLIB set**
  - Known to be difficult for MIP solvers
  - Sub MIP solves quickly
  - Big speedups on had, nug QAPs of dimension 12
  - Solved had, nug QAPs of dimension 14 and 16 that regular Gurobi cannot solve with defaults or parameter tuning
  - Doesn’t scale up to larger QAPs
    - Sub MIP cuts require basic linearization
    - Number of bilinear terms grows quadratically as dimension increases
    - Node throughput slows dramatically as problem size increases
- **Proprietary models from a prospect**
  - SubMIP solves remained far from optimality after over an hour
  - Extended BQP cuts in 9.1.x and beyond were very effective

Reformulations

GUROBI OPTIMIZATION

- Back to the p-Dispersion-Sum problem
- Cardinality based cut enabled Gurobi to solve the n=50, k=10 instance within 30 seconds
- But how well does it scale?
- An instance with n=100, k= 20 and the cardinality cut did not solve to optimality in several hours, despite better gap
- Instead of maximizing the sum of distances, maximize the minimum distance among the k chosen points (and k*(k-1)/2 associated distances):

$$\boldsymbol{Max\ \Delta}$$

$$s.t. \Delta \leq \boldsymbol{d_{ij}} + \boldsymbol{M}(\mathbf{1} - \boldsymbol{x_i} * \boldsymbol{x_j}) \qquad // \boldsymbol{M} = \mathbf{max}\, \boldsymbol{d_{ij}}$$

Only binding when $x_i = x_j$=1

$$\sum_{j=1}^{n} \boldsymbol{x_j} = \boldsymbol{k}$$

$$\boldsymbol{x_j} \in \{\mathbf{0}, \mathbf{1}\}$$

(source: http://yetanothermathprogrammingconsultant.blogspot.com/2019/06/maximum-dispersion.html)

Reformulations

Only binding when $x_i = x_j$=1

- $\boldsymbol{Max\ \Delta}$

  $$\boldsymbol{s.t.\ \Delta \le d_{ij} + M(1 - x_i * x_j)} \quad \Longleftrightarrow \quad \boldsymbol{\Delta \le d_{ij} + M(1 - x_i) + M(1 - x_j))}$$

  $$\boldsymbol{\sum_{j=1}^{n} x_j = k}$$

  $\boldsymbol{x_j \in \{0, 1\}}$ **(Formulation 1)** **(Formulation 2)**

- **Formulation 1 (n=100, k=20)**
  - Defaults (PreQLinearize=1): 16 seconds
  - PreQLinearize=0: Timed out after 3 hours (solving nonconvex MIQCP)

- **Formulation 2 (n=100, k=20)**
  - Defaults: 2 seconds

(source: http://yetanothermathprogrammingconsultant.blogspot.com/2019/06/maximum-dispersion.html)

A few loose ends

- **The subMIP and cardinality cut approaches required direct access to the linearization variables**
  - Can generate and work on the presolved model
  - Easier to just do the simple linearization (PreQLinearize=1) yourself

```
def qlinearize(model, useobj, zvars = None):
    qobj     = model.getObjective()
    t        = model.ModelSense
    linpart  = qobj.getLinExpr()
    model.setObjective(linpart)
    dupdict  = {}
    for k in range(qobj.size()):
        qcoef    = qobj.getCoeff(k)
        xi       = qobj.getVar1(k)
        xj       = qobj.getVar2(k)
        key1     = (xi, xj)
        key2     = (xj, xi)
        if key1 in dupdict or key2 in dupdict:
            continue        # this product already linearized; don't duplicate
        else:
            dupdict[key1] = 1   # first time this xi, xj pair encountered.
            dupdict[key2] = 1   # proceed with linearization.
        qcoeff   = qobj.getCoeff(k)
        zname    = "z_" + xi.VarName + "_" + xj.VarName
        zij      = model.addVar(obj=qcoeff, vtype=GRB.BINARY, name = zname)
        suffix   = xi.VarName + "_" + xj.VarName
        skip1    = False
        skip2    = False
        if useobj:
            down = qcoef*t > 0.0
            if down:
                skip1 = True
            else:            # qobj only contains nonzero Q elements
                skip2 = True

        if not skip1:
            cname = "lin1_"+ suffix
            model.addConstr(zij - xi <= 0, name = cname)
            cname = "lin2_"+ suffix
            model.addConstr(zij - xj <= 0, name = cname)
        if not skip2:
            cname = "lin3_"+ suffix
            model.addConstr(xi + xj - zij <= 1, name = cname)
        if zvars != None:
            zvars.append(zij)
```

(appendix of this deck contains this code in a size that can actually be read)

A few loose ends

- **Sometimes better to generate all 3 constraints associated with PreQLinearize=1 instead of the 1 or 2 that Gurobi generates**
  - Not needed for correctness, but can yield tighter relaxations
  - Under investigation to try to get the tighter formulation without the additional constraint(s)

A few loose ends

- **MILPs that are actually MINLPs in disguise**
  - We already saw this with neos-911970
    - Soft knapsack constraints were a multinomial objective in disguise
  - Don't necessarily want to reformulate the MILP into the MINLP
    - But do want to consider both formulations, consider anything in the unused formulation that will help the other run faster (e.g. create the Padberg Graph)
  - Other examples
    - Overlap or interference conditions
    - Logical conditions (e.g. SAT models)
      - z variable models an and for the two binary x variables
    - The open MIPLIB model neos-2629914-sudost
  - Solve the MILP or MINLP, but use info from both to improve performance

$$z_{ij} = x_i x_j$$

$$z_{ij} \le x_i$$

$$z_{ij} \le x_j$$

$$z_{ij} \ge x_i + x_j - 1$$

Key Takeaways

- **Non Default parameter settings can help**
  - PreQLinearize
  - RLT, BQP and ZeroHalf cuts
  - Don't forget the NodeMethod parameter to improve node LP solve times
- **When tightening the formulation, need to consider the linearization variables**
  - Linearized model offers more opportunities to tighten than convexified model
    - Albeit with potentially slower node throughput
  - Gurobi's linearizations emphasize compact formulations
    - Sometimes larger, basic linearization may work better
    - Easy to implement (see Appendix)
  - RLT type strategy: Multiply linear constraints by binary variable, linearize and combine constraints
  - Padberg graph can yield insights
  - Solve subMIPs including the linearization variables to determine bounds on the number of linearization variables that must be 1
- **Look for reformulation opportunities**
  - Make sure quadratic conditions involving binaries are really quadratic.
  - MINLP formulation in disguise can be used to tighten MILP formulation being used

References and Resources

GUROBI OPTIMIZATION

1. Dispersion Problems (Kalvelagen): http://yetanothermathprogrammingconsultant.blogspot.com/2019/06/maximum-dispersion.html
2. Products of binaries (Achterberg): https://www.gurobi.com/resource/products-of-variables-in-mixed-integer-programming/
3. More on the Padberg Graph and products of binaries (Klotz): http://orwe-conference.mines.edu/files/IOS2018SpatialPerfTuning.pdf
4. Padberg's original Boolean Quadric Polytope Paper: https://link.springer.com/article/10.1007/BF01589101
5. Reformulating IPs as QUBOs (Glover, Kochenberger, Du)

   https://www.springerprofessional.de/en/quantum-bridge-analytics-i-a-tutorial-on-formulating-and-using-q/17436666
6. Other literature on linearizations of products of binaries: https://www.hindawi.com/journals/jam/2020/5974820/

Thank You

Questions?

Appendix

- **Some sample code using the Gurobi Python API to do the basic linearization**

```
def qlinearize(model, useobj, zvars = None):
    qobj     = model.getObjective()
    t        = model.ModelSense
    linpart  = qobj.getLinExpr()
    model.setObjective(linpart)
    dupdict  = {}
    for k in range(qobj.size()):
        qcoef    = qobj.getCoeff(k)
        xi       = qobj.getVar1(k)
        xj       = qobj.getVar2(k)
        key1     = (xi, xj)
        key2     = (xj, xi)
        if key1 in dupdict or key2 in dupdict:
            continue        # this product already linearized; don't duplicate
        else:
            dupdict[key1] = 1   # first time this xi, xj pair encountered.
            dupdict[key2] = 1   # proceed with linearization.
        qcoeff   = qobj.getCoeff(k)
        zname    = "z_" + xi.VarName + "_" + xj.VarName
        zij      = model.addVar(obj=qcoeff, vtype=GRB.BINARY, name = zname)
```

Appendix

- Some sample code using the Gurobi Python API to do the basic linearization

```
suffix   = xi.VarName + "_" + xj.VarName
skip1    = False
skip2    = False
if useobj:
    down = qcoef*t > 0.0
    if down:
        skip1 = True
    else:             # qobj only contains nonzero Q elements
        skip2 = True

if not skip1:
    cname = "lin1_"+ suffix
    model.addConstr(zij - xi <= 0, name = cname)
    cname = "lin2_"+ suffix
    model.addConstr(zij - xj <= 0, name = cname)
if not skip2:
    cname = "lin3_"+ suffix
    model.addConstr(xi + xj - zij <= 1, name = cname)
if zvars != None:
    zvars.append(zij)
```