1000X MIP Tricks

Bob Bixby

12 June 2012, Bill Cunningham's 65th

Gurobi Optimization

Reminiscences on Matroids

A Characterization of Ternary Matroids

Gurobi Optimization


PREPARATION H
HEMORRHOIDAL OINTMENT
Prevents Further Irritation
• Prompt Soothing Relief from Painful Burning, Itching and Discomfort
• Shrinks Swollen Hemorrhoidal Tissue
• Protects Irritated Tissue
• Relieves Internal and External Discomfort
NET WT 1 OZ (28g)

Outline

- Introduction
  - Progress in Solving Mixed-Integer Programs
- MIP tricks
  - Knapsack
  - Implied integer
  - Disjoint subtrees
  - Modular inverse reduction
  - Markshare

Gurobi Optimization

A Definition

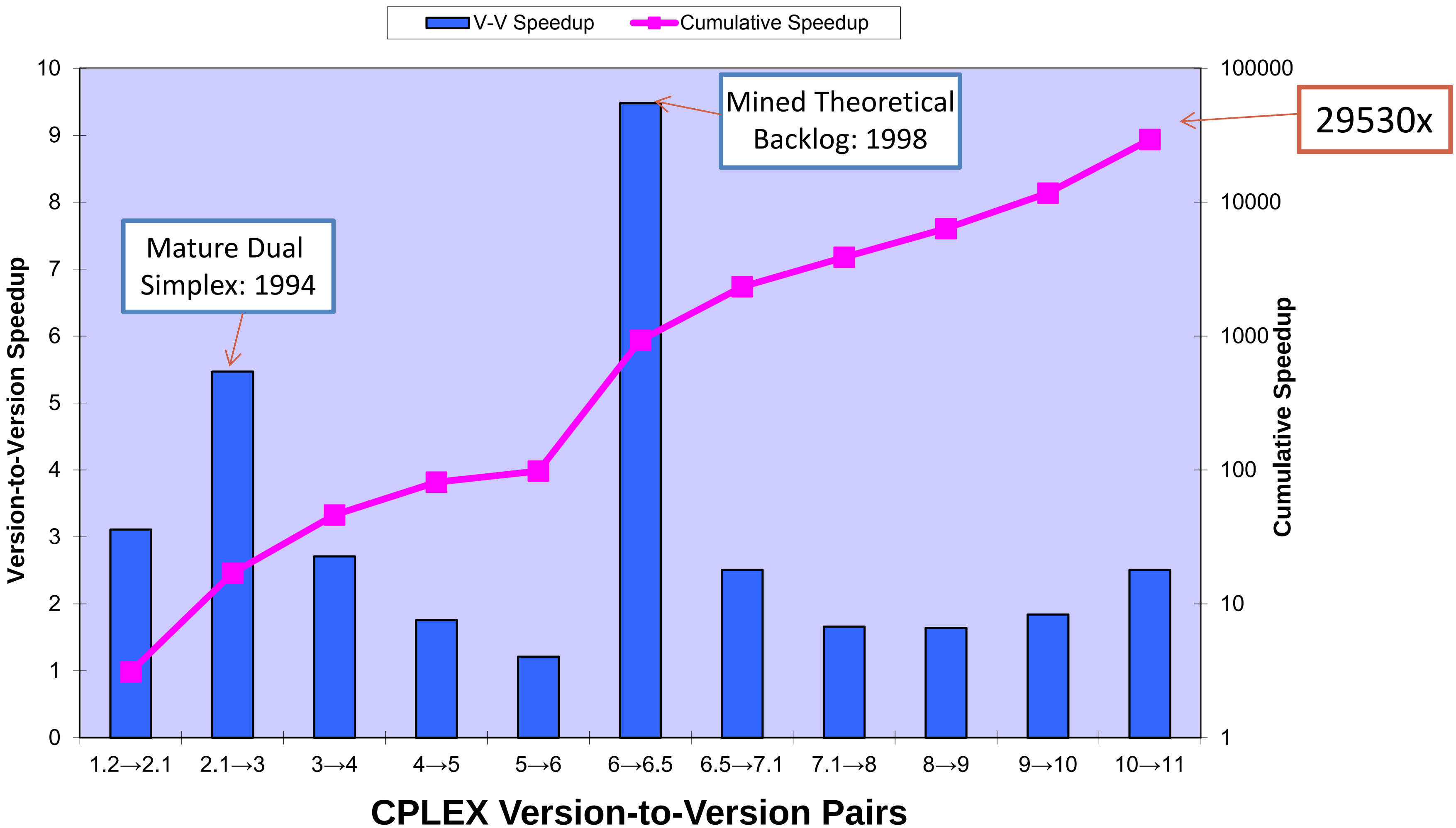
A *mixed-integer program* (MIP) is an optimization problem of the form

$$\boxed{\begin{array}{ll} \textit{Minimize} & c^T x \\ \textit{Subject to} & Ax = b \\ & l \le x \le u \\ \multicolumn{2}{l}{\text{some or all } x_j \text{ integer}} \end{array}}$$

Gurobi Optimization

Computational Progress in Mixed-Integer Programming: 1991-Present

Speedups 1991-2007



Gurobi Solver: Version 1.0 Released May 2009

- Public benchmarks showed that CPLEX 11.0 and Gurobi 1.0 were roughly equivalent

- Gurobi: Version-to-version improvements:
  - Gurobi 1.0 -> 2.0: 2.2X
  - Gurobi 2.0 -> 3.0: 2.9X (6.4X)
  - Gurobi 3.0 -> 4.0: 1.3X (8.3X)
  - Gurobi 4.0 -> 5.0: 1.9X (16.2X)

Gurobi Optimization

Overall MIP Improvement: 1991–Present

- Algorithmic Improvement
  - Factor 29530 x 16.2: 475,000x speedup
  - Like investing $1 at 92% annual interest for 20 years.
- Machine Improvement
  - Factor 2,000x speedup
- Total Improvement
  - Factor ~10^9x speedup

Gurobi Optimization

CPLEX 6.5 – 1997/98:
The Breakthrough

Gurobi Optimization

Computational Results III: 78 Models

Before CPLEX 6.5 – not solvable
After CPLEX 6.5 – solvable < 1000 seconds

Cutting planes	33.3x
Presolve	7.7x
Variable selection	2.7x
Node presolve	1.3x
Heuristics	1.1x
Dive probing	1.1x

Gurobi Optimization

Gurobi MIP Solver

Gurobi Optimization

Gurobi MIP Solver

- Branch-and-cut algorithm
- Deterministic shared memory parallel
- Key building blocks
  - Dual simplex
  - Cut planes
  - Heuristics for finding integer-feasible solutions
  - Presolve
  - Branch variable selection
- Many tricks

Cutting Planes

- Gomory
- Knapsack cover
- Flow cover
- GUB cover
- MIR
- Clique
- Implied bounds
- Zerohalf
- Mod-k
- Network
- Submip

Heuristics

- Rounding
- RINS
- Solution improvement
- Feasibility pump
- Diving
- Alternative optimal solutions with less integer infeasibility
- Etc.

Presolve

- Bound strengthening
- Row “analysis”
- Coefficient reduction
- Aggregation
- Clique generation
- Probing
- Etc.

Gurobi Optimization

Variable Selection Technology

- Pseudo costs
- Strong branching
- Reliability branching
- Etc.

MIP Tricks

Gurobi
Optimization

Knapsacks

- P2m2p1m1p0n100
  - A 0-1 knapsack in MIPLIB 2010 infeasible set
  - 100 binary variables (if slack isn't counted)
  - rhs = 80424
- Solutions times (from Mittelmann)
  - Gurobi 4.6: 0.06 sec
  - CPLEX: 12.3: 671 sec, 12.4: 2.36 sec
  - XPRESS 7.2.1: 1961sec
- Our trick
  - Run branch-and-cut for some number of nodes
  - Check whether it is a special MIP model, like knapsack
  - Use virtual time (deterministic) spent on B&C vs. estimate time of dynamic programming (here O(n*rhs))
  - Use dynamic programming to solve it

Gurobi Optimization

Implied Integer, Example

- Model b_ball (first version of MIPLIB 2010)

Max x12

S.t. x12 – x1 <= 0

....

x12 – x11 <= 0

2 x1 – x13 – x14 – x15 – x16 – x17 – x18 – x19 – x20 = 0

....

2 x11 – x93 – x94 – x95 – x96 – x97 – x98 – x99 – x100 = 0

x13 + x21 + x29 + x37 + x45 + x53 + x61 + x69 + x77 + x85 + x93 = 5

....

x20 + x28 + x36 + x44 + x52 + x60 + x68 + x76 + x84 + x92 + x100 = 5

x1, …, x12 are continuous

x13, …, x100 are binary

Gurobi Optimization

Implied Integer, Example

- It is easy to see
  - 2 x1, …, 2 x11 must take integer values
  - Hence 2 x12 will take integer values
  - Obj. gcd is 0.5
- The trick of recognizing obj. gcd =0.5 reduces the solution time from 10000+ seconds to 0.01 second

Gurobi Optimization

Implied Integer

- The trick can be extended to catch many more cases
- Recognizing implied integer variables for cuts, bound strengthening, obj. gcd and etc. is very useful and has significant impact on overall performance

Gurobi Optimization

Disjoint Subtrees

- Basic principle of branching:
  - Feasible regions for child nodes after a branch should be disjoint
- Not always the case
- Simple example – integer complementarities:
  - $x \leq 10\ b$
  - $y \leq 10\ (1-b)$
  - x, y non-negative ints, $x \leq 10$, $y \leq 10$, b binary
  - Branch on b: $x=y=0$ feasible in both children

Gurobi Optimization

Recognizing Subtree Overlap

- Problem arises when sole purpose of branching variable is to bound other variables
  - Otherwise, b=0/b=1 split is typically sufficient to make the subtrees disjoint
- Recognizing overlap:
  - Constraints involving branching variable must be redundant after branch
  - Domains of remaining variables must overlap

Removing Overlap

- Simplest way to remove overlap:
  - Modify variable bound in one subtree
- Integer complementarities example:
  - $x \leq 10\, b$
  - $y \leq 10\, (1-b)$
  - Branch on b: $x=y=0$ feasible in both children
- $b=0$ child: $x = 0,\ 10 \geq y \geq 0$
- $b=1$ child: $y = 0,\ 10 \geq x \geq 1$

Gurobi Optimization

Performance Impact

- Overlap present in several models
  - 35 out of 510 models in our test set
- Performance impact can be huge
  - Model neos859080 goes from 10000+ seconds to 0.01s
  - Makes it tough to quote mean improvements over a small set
- Median improvement for affected models is ~1%

Gurobi Optimization

Modular Inverse Reduction

- Consider
  - $a\, x + b\, y = c$
  - x, y are integer variables
  - a, b and c are integers, $a > 1$
  - Assume $\gcd(a,b) = 1$
    - Otherwise a Euclidean reduction is possible
  - Observation: Then x(mod b) and y(mod a) are constants.
- Reduction
  - Substitute $y = a\, z + d$, where d can be computed by modular multiplicative inverse
  - z has a smaller search space than x and y
- General application
  - Can easily be extended to general “all integer” constraints.

Gurobi Optimization

Modular Inverse Reduction

- Simplex example
  - Min $x + y$
    s.t. $1913\ x + 1867\ y = 3618894$
    $x, y \geq 0$, are integral variables
- Reduction
  - Using modular inverse, you get $y = 1913\ z + 1009$, with $z \geq 0$
  - So $1913\ x + 3571571\ z = 1735091$, or
    $x + 1867\ z = 907$
  - With the reductions, presolve solves it, while without the reduction it takes 1942 nodes.

Gurobi Optimization

Performance Impact

- Less than 3% models are affected
- Performance impact can be huge
  - A model from GAMS goes from 10000+ seconds to 0.05 seconds
  - Overall impact is positive, but small

Gurobi Optimization

Markshare Models

- Models
  - Less than 100 binary variables
  - Less than 7 knapsacks
  - Minimize sum of slacks
- MIPLIB
  - markshare1 and markshare2 in MIPLIB 2003
  - markshare_5_0 in MIPLIB 2010
- Cornuejols, Dawande 1998
  - Use basis reduction to solve
  - Branch-and-cut fails to solve markshare1 and 2

Gurobi Optimization

Markshare Model: Markshare_5_0

Minimize

s1 + s2 + s3 + s4 + s5

Subject To

C1_: s1 + 17 x1 + 75 x2 + 9 x3 + 87 x4 + 58 x5 + 79 x6 + 69 x7 + 37 x8 + 88 x9 + 75 x10 + 45 x11
+ 35 x12 + 73 x13 + 26 x14 + 39 x15 + 78 x16 + 85 x17 + 58 x18 + 72 x19 + 8 x20 + 46 x21
+ 11 x22 + 55 x23 + 39 x24 + 57 x25 + 96 x26 + 87 x27 + 16 x28 + 27 x29 + 26 x30 + 93 x31
+ 44 x32 + 79 x33 + 12 x34 + 8 x35 + 95 x36 + 2 x37 + 15 x38 + 38 x39 + 15 x40 = 987

C2_: s2 + 53 x1 + 88 x2 + 43 x3 + 26 x4 + 31 x5 + 77 x6 + 10 x7 + 77 x8 + 71 x9 + 22 x10 + 76 x11
+ 41 x12 + 65 x13 + 93 x14 + 50 x15 + 69 x16 + 44 x17 + 61 x18 + 58 x19 + 63 x20 + 46 x21
+ 63 x22 + 13 x23 + 97 x24 + 14 x25 + 45 x26 + 32 x27 + 96 x28 + 36 x29 + 40 x30 + 10 x31
+ 96 x32 + 99 x33 + 58 x34 + 87 x35 + 15 x36 + 91 x37 + 65 x38 + 6 x39 + 96 x40 = 1111

C3_: s3 + 97 x1 + 79 x2 + 81 x3 + 57 x4 + 28 x5 + 97 x6 + 58 x7 + 44 x8 + 37 x9 + 93 x10 + 2 x11
+ 77 x12 + 73 x13 + 59 x14 + 43 x15 + 64 x16 + 75 x17 + 6 x18 + 5 x19 + 78 x20 + 71 x21
+ 12 x22 + 30 x23 + 7 x24 + 69 x25 + 36 x26 + 73 x27 + 19 x28 + 15 x29 + 16 x30 + 84 x31
+ 55 x32 + 32 x33 + 53 x34 + 43 x35 + 21 x36 + 73 x37 + 59 x39 + 48 x40 = 984

C4_: s4 + 94 x1 + 76 x2 + 12 x3 + x4 + 50 x5 + 85 x6 + 86 x7 + 9 x8 + 86 x9 + 79 x10 + 58 x11
+ 10 x12 + 83 x13 + 75 x14 + 91 x15 + 51 x16 + 89 x17 + 97 x18 + 57 x19 + 47 x20 + 42 x21
+ 65 x22 + 88 x23 + 59 x24 + 22 x25 + 100 x26 + 16 x27 + 70 x28 + 70 x29 + 99 x30 + 65 x31
+ 66 x32 + 85 x33 + 68 x34 + 97 x35 + 33 x36 + 80 x37 + 16 x38 + 87 x39 + 60 x40 = 1262

C5_: s5 + 42 x1 + 99 x2 + 87 x3 + 46 x4 + 24 x5 + 85 x6 + 85 x7 + 74 x8 + 13 x9 + 48 x10 + 79 x11
+ 50 x12 + 57 x13 + 44 x14 + 3 x15 + 33 x16 + 43 x17 + 58 x18 + 8 x19 + 68 x20 + 59 x21
+ 23 x22 + 75 x23 + 96 x24 + 87 x25 + 7 x26 + 54 x27 + 38 x28 + 72 x30 + 5 x31 + 2 x32 + 76 x33
+ 63 x34 + 94 x35 + 55 x36 + 41 x37 + 39 x38 + 19 x39 + 31 x40 = 991

Gurobi Optimization

Markshare Model

- Simple example
  - $6\ x_1 + 7\ x_2 + 7\ x_3 + \ldots + 7\ x_{29} + 8\ x_{30} + s_1 = 29$
    $8\ x_1 + 7\ x_2 + 7\ x_3 + \ldots + 7\ x_{29} + 6\ x_{30} + s_2 = 29$
  - Let $f_k(u)$ = first i, $\Sigma_{\{1 \leq j \leq i\}} a_{kj}\ x_j = u$ is feasible
    - $f_1(6) = 1$, $f_1(13) = 2$, $f_1(14) = 3$, $f_1(20) = 3$, …, $f_1(29) = 30$
    - $f_2(8) = 1$, $f_2(15) = 2$, $f_2(14) = 3$, $f_2(22) = 3$, $f(29) = 4$, …, $f_2(23) = \text{inf}$
  - Try $s_1 = 0$, $s_2 = 0$
    - Backwards, start with x_{30}.
    - If $x_{30} = 0$, then rhs's remain 29, but $f_1(29) = 30$, so the first constraint is infeasible
    - If $x_{30} = 1$, then for the second constraint, rhs – 6 = 23, but $f_2(23) = \text{inf}$, so it is infeasible
    - It is infeasible for $s_1 = 0$, $s_2 = 0$
  - Cost to compute $f_k(u)$ is O(rhs*n)

Gurobi Optimization

Markshare Models: Our Trick

- Dynamic programming plus enumerating
  - Combine 2 to 3 constraints, say 2, and compute $f(u_1, u_2)$ = first i, $\Sigma_{\{1 \leq j \leq i\}} a_{kj} x_j = u_k$ is feasible
    Operations $O(n \times b_1 \times b_2)$
  - Try $\Sigma s_k = 0$; $\Sigma s_k = 1$, $s_k = 1$, $k = 0, 1, \ldots$
  - Backward Looping over x_j, $j = n, \ldots, 1$
    - At j=i, let

      $x_j = v_j$, for $j = n, \ldots, i$

      $u_k = b_k - \Sigma_{\{i \leq j \leq n\}} a_{kj} v_j$
    - If $f(u_1, u_2) \geq i$, $x_j = v_j$, for $j = n, \ldots, i$, is infeasible, no need to continue to enumerate x_j, for $j < i$

Gurobi Optimization

Markshare Models: Computation

- Trick is implemented in Gurobi 4.5
- Solution times on i7-920, threads=4

	Gurobi 4.0	Gurobi 4.6
Markshare_5_0	1347s	0.74s
Markshare1	>7200s	243s
Markshare2	>7200s	5958s

Gurobi Optimization

Conclusions

- MIP tricks
  - A lot of them are easy to find by just staring at models and often are also easy to apply
  - Many of them are quite effectively on a small fraction of models
  - An interesting challenge for combinatorial mathematicians

- Finding MIP tricks is always fun!

Thank You

Gurobi

Optimization