Introduction to Tuning

Topics

- Performance issues
- The Gurobi log file
- Performance tuning
  - Gurobi tuning tool
  - Performance guidelines for
    - General issues
    - Presolve
    - Continuous optimization
    - Integer optimization
  - Unexpected behavior

GUROBI OPTIMIZATION

Performance Issues

Causes of slow performance

- Numerical issues
- Exhausting memory
- Unrealistic tolerance values
- A large or difficult model

Troubleshooting slow performance

- Determine what cases cause the slow performance
- Determine what algorithmic part is the bottleneck

- Common support case
  - “Gurobi is soooo slow”
  - Reason: Model building outside of Gurobi takes 30 minutes
  - Model solving takes only a few seconds

- Parameter tuning tool
- Performance guidelines
- Send test models and log files to Gurobi

What cases cause slow performance

- What **data** cause slow performance
  - Every case or selected conditions

- Is the slow performance predictable or seemingly random (**reproducibility**)

- Is it comparably slow on different computers
  - Gurobi can provide temporary licenses for testing on other computers
  - Try our Cloud

What algorithmic part is the bottleneck

- Model initialization and solution retrieval
  - Test MPS file using `gurobi_cl`
  - See if solution times are much faster

- Solve times
  - Presolve
  - Solving (initial LP)
  - At node 0 of MIP
  - Other nodes of MIP
  - Log shows time spent in presolve, LP relaxation, MIP root, nodes

- Use the logs to identify the bottleneck

GUROBI OPTIMIZATION

The Gurobi Log File

Demonstration – Solving from a file

- Solve the model in the Gurobi Interactive Shell
  - Start the Interactive Shell Type commands:

```
help(read)
m = read('c:/Users /<your user name>/ Desktop/bell4.mps')
m.optimize()
m.reset()
m.Params.threads = 1
m.optimize()
```

- Solve the model with the Command Line Tool
  - Start a terminal window
  - Type commands:

```
gurobi_cl c:/Users/<your user name>/Desktop/bell4.mps
gurobi_cl Threads=1 c:/Users/<your user name>/Desktop/bell4.mps
```

Gurobi MIP log - header

```
Gurobi Optimizer version 7.0.2 build v7.0.2rc0 (mac64)
Copyright (c) 2017, Gurobi Optimization, Inc.

Read MPS format model from file bell4.mps
Reading time = 0.06 seconds
BELL4: 105 rows, 117 columns, 302 nonzeros
Optimize a model with 105 rows, 117 columns and 302 nonzeros
Coefficient statistics:
  Matrix range     [8e-05, 1e+03]
  Objective range  [7e-03, 6e+04]
  Bounds range     [1e+00, 1e+04]
  RHS range        [1e+00, 1e+04]
Found heuristic solution: objective 1.89378e+07
Presolve removed 32 rows and 29 columns
Presolve time: 0.01s
Presolved: 73 rows, 88 columns, 226 nonzeros
Variable types: 37 continuous, 51 integer (25 binary)
```

Gurobi MIP log – root solve

```
Root relaxation: objective 1.850662e+07, 65 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1.8507e+07    0   22 1.8938e+07 1.8507e+07  2.28%     -    0s
H    0     0                       1.878928e+07 1.8507e+07  1.50%     -    0s
H    0     0                       1.873790e+07 1.8507e+07  1.23%     -    0s
H    0     0                       1.858818e+07 1.8507e+07  0.44%     -    0s
     0     0 1.8516e+07    0   18 1.8588e+07 1.8516e+07  0.39%     -    0s
H    0     0                       1.857897e+07 1.8516e+07  0.34%     -    0s
     0     0 1.8517e+07    0   18 1.8579e+07 1.8517e+07  0.34%     -    0s
     0     0 1.8517e+07    0   19 1.8579e+07 1.8517e+07  0.33%     -    0s
     0     0 1.8517e+07    0   21 1.8579e+07 1.8517e+07  0.33%     -    0s
     0     0 1.8517e+07    0   21 1.8579e+07 1.8517e+07  0.33%     -    0s
     0     2 1.8517e+07    0   21 1.8579e+07 1.8517e+07  0.33%     -    0s
```

Gurobi MIP log – search tree exploration

GUROBI OPTIMIZATION

```
     Nodes    |    Current Node    |     Objective Bounds      |     Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

...
H  253    36                       1.857469e+07 1.8519e+07  0.30%   1.9    0s
H  702   123                       1.855884e+07 1.8519e+07  0.22%   1.9    0s
H  714   111                       1.854998e+07 1.8519e+07  0.17%   1.8    0s
H  721   111                       1.854989e+07 1.8519e+07  0.17%   1.9    0s
H 2824   585                       1.854852e+07 1.8521e+07  0.15%   1.9    0s
H 2867   558                       1.854752e+07 1.8521e+07  0.14%   1.9    0s
* 2877   557              37       1.854744e+07 1.8521e+07  0.14%   1.9    0s
H 3745   526                       1.854362e+07 1.8536e+07  0.04%   2.0    0s
* 3757   489              43       1.854353e+07 1.8536e+07  0.04%   2.0    0s
H 4156   266                       1.854322e+07 1.8537e+07  0.03%   2.0    0s
* 4167   230              41       1.854306e+07 1.8537e+07  0.03%   2.0    0s
H 4426   165                       1.854302e+07 1.8537e+07  0.03%   1.9    0s
H 5293    26                       1.854204e+07 1.8538e+07  0.02%   1.9    0s
* 6423     6              39       1.854196e+07 1.8540e+07  0.01%   1.8    0s
H 6650    49                       1.854189e+07 1.8540e+07  0.01%   1.8    0s
* 6662    47              38       1.854183e+07 1.8540e+07  0.01%   1.8    0s
```

Gurobi MIP log – summary

```
Cutting planes:
  Gomory: 24
  MIR: 14

Explored 6835 nodes (12379 simplex iterations) in 0.24 seconds
Thread count was 8 (of 8 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 1.854182583800e+07, best bound 1.853999618018e+07, gap 0.0099%
```

GUROBI OPTIMIZATION

Performance Tuning

Parameter tuning basics

- Parameter changes can have a big effect
  - http://www.gurobi.com/documentation/current/refman/parameters.html

- 57 Gurobi parameters affect performance
  - Too many to try by hand
  - Often not obvious which ones to try first

- Follow basic rules of thumb
  - Less is more
  - Always try primal, dual and barrier for an LP
  - Don't change inapplicable parameters
    - Ex: don't set barrier parameters if you use simplex
  - If presolve takes most of the runtime, try Presolve=1

grbtune: parameter tuning tool

- Automatically finds sets of parameters that give good performance
- Two modes
  - Fastest time to optimality
  - Smallest MIP gap in a fixed amount of time
- Can run distributed across multiple computers
- An API is available
- For robust results, tune with a representative set of models

- Examples:

```
grbtune TuneTimeLimit=3600 modelA1.mps modelA2.mps modelA3.mps
grbtune TuneTimeLimit=7200 TimeLimit=300 modelB1.mps modelB2.mps
```

Demonstration: parameter tuning tool

GUROBI OPTIMIZATION

- Apply tuning with default settings
  - Start the Interactive Shell (double-click on icon)
    - Type commands:

```
m = read('c:/Users/<your user name>/Desktop/misc07.mps')
m.tune()
```

  - Use Command Line Tool:
    - Start a terminal window
    - Type command:

```
grbtune c:/Users/<your user name>/Desktop/misc07.mps
```

- Note:
  - Often there's no replacement for brute-force search
  - More than 2X improvement from tuning
  - Most important parameter:
    - VarBranch=1

Tuning tool output

GUROBI OPTIMIZATION

- Tries multiple parameter combinations, looking for improving set...

```
Testing candidate parameter set 16...

        MIPFocus 2
        VarBranch 1

Solving with random seed #1 ... runtime 3.01s+

Progress so far: baseline runtime 3.91s, best runtime 1.83s
Total elapsed tuning time 99s (18s remaining)
```

- Reports best results found when it finishes…

```
Tested 20 parameter sets in 117.17s

Baseline parameter set: runtime 3.91s

…

Improved parameter set 3 (runtime 2.52s):

        VarBranch 1
```

Performance guidelines by category

- General issues
  - Model initialization
  - Numerical issues
  - Memory
- Presolve
- Continuous optimization
- Integer optimization

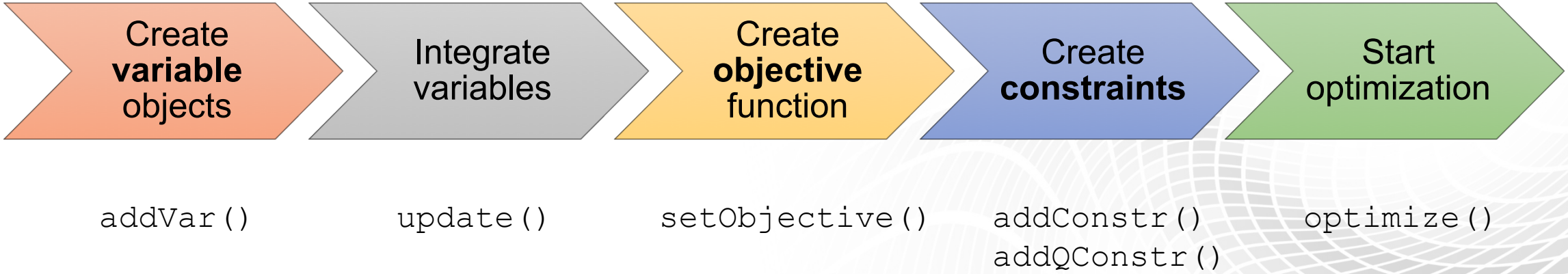
GUROBI OPTIMIZATION

General Issues

Model initialization

- Each matrix generator has its own pitfalls and best practices
  - Use iterators effectively for your API

- Look for: bottleneck via a code profiler or measure yourself
- OO interfaces are thin layer upon C matrix interface
- With Gurobi OO interfaces, take advantage of lazy updates
  - Only call `update` function when you need to reference new objects



Manage model objects

- C++ considerations
  - Always pass by reference, not by value
  - When using pointers to `GRBModel` and `GRBEnv`, delete when finished
  - Reuse an existing object (`sum += x`) instead of creating a new one (`sum = sum + x`)

- Java and .NET considerations
  - Garbage collector typically does not free `GRBEnv` quickly
  - Call the `Dispose()` methods on the `GRBModel` and `GRBEnv` objects when finished

Memory

GUROBI OPTIMIZATION

- Insufficient memory can destroy performance
  - Virtual memory via disk is far slower than RAM
  - Parallel optimization requires more memory
- Look for: memory use via system monitor tools on computer
  - Ex: System Monitor, top, Activity Monitor
- Helpful parameters
  - Decrease `Threads`
  - Set `NodefileStart` to store MIP node info on disk
    - Only helpful when solving a MIP that requires many nodes!
- Memory is cheap; no need to skimp

GUROBI OPTIMIZATION

Presolve

Presolve

- Tradeoff: spend time up front with hope of simplifying model
- Look for: performance with different presolve parameters
- Primary control: `Presolve` parameter
  - Reduce if spending too much time up front
  - Increase to hope to get a simpler model
- Additional parameters for fine-grain control
  - `PrePasses`
  - `Aggregate`
  - `AggFill`
  - `PreSparsify`
  - `PreDual`
  - `PreDepRow`

GUROBI OPTIMIZATION

Continuous Optimization

Continuous algorithms

- Dual simplex
- Primal simplex
- Barrier

- Concurrent (LP)
  - Use multiple algorithms at the same time on multiple processor cores
  - Deterministic and non-deterministic versions available
  - Multiple algorithms makes it very robust
  - Requires more memory

Defaults for continuous optimization

- LP Concurrent (non-deterministic)
- QP Barrier
- MIP root Dual simplex or (deterministic) concurrent, depending on model size
- MIP nodes Dual simplex

- Parameters used to select the algorithm
  - `Method`: continuous models and root of MIPs
  - `NodeMethod`: nodes of MIPs

The fastest solver

- Interesting:
  - Concurrent LP performance worse than dual simplex performance
  - Reason?

- Concurrent fastest on average

- Concurrent never fastest for a specific model
  - If barrier wins:
    - Concurrent wasted one thread on dual
  - If dual wins:
    - Dual had to fight for resources with barrier
      - Most limiting resource: the cooling fan!

LP – example 1

- First run

```
gurobi> m.optimize()
...
Solved with dual simplex
Solved in 11615 iterations and 3.72 seconds
Optimal objective  2.382165864e+10
gurobi> print m.getVars()[0].X
351.0
```

- Second run of same model

```
gurobi> m.optimize()
...
Solved with barrier
Solved in 53305 iterations and 3.70 seconds
Optimal objective  2.382165864e+10
gurobi> print m.getVars()[0].X
0.0
```

LP – example 1

- Default solver is non-deterministic concurrent
- Different optimal basis possible when dual and barrier runtimes are very close
  - Fortunately, this is rare

- If this is an issue, use a deterministic method
  - Deterministic concurrent (will be a bit slower)
  - Parallel barrier
  - Simplex

LP – example 2

- Small LP solves quickly
- Larger LP solves much more slowly

- "Is the disk activity light flashing on your PC?"
  - "Yes, why do you ask?"

Memory usage

- Concurrent uses much more memory than dual
  - Concurrent invokes barrier
    - Barrier typically uses a lot more memory than dual
  - Concurrent runs multiple solvers at once
    - Each needs a copy of the model

- If memory is tight, use dual simplex

Notable continuous parameters

- `NormAdjust`
  - Select different simplex pricing norm variants
- `SimplexPricing`
  - Simplex variable pricing strategy
- `Crossover`
  - Determines strategy used to produce basis from initial crossover basis
- `CrossoverBasis`
  - Determines strategy used to produce initial basis from barrier result

GUROBI OPTIMIZATION

Integer Optimization

Why is your MIP difficult

- Time to solve LP/QP relaxations?
- Moving the bound?
- Finding feasible solutions?

If relaxations are the bottleneck

- Use tuning methods for continuous optimization
  - Try different methods for root and nodes
  - Check for memory issues
  - Try different values of `NormAdjust` parameter

MIP – example 1

- MIP log looks like this…

```
     Nodes    |    Current Node    |     Objective Bounds      |     Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 -264137.12    0   75          - -264137.12     -      -    0s
H    0     0                       -0.0000107 -264137.12     -      -    0s
H    0     0                      -162837.1173 -264137.12  62.2%     -    0s
     0     0 -251769.54    0   50 -162837.12 -251769.54  54.6%     -    0s
     0     0 -246602.68    0   83 -162837.12 -246602.68  51.4%     -    0s
     0     0 -241768.49    0   29 -162837.12 -241768.49  48.5%     -    0s
H    0     3                      -180084.9071 -241768.49  34.3%     -    0s
     0     3 -241768.49    0   29 -180084.91 -241768.49  34.3%     -    0s
* 2343  1391              49     -181346.8006 -214776.82  18.4%   6.4    2s
* 2440  1339              70     -183734.9437 -214277.60  16.6%   6.4    2s
* 3103  1527              59     -183933.1807 -213309.24  16.0%   6.4    2s
* 3710  1825              61     -184549.0628 -212449.79  15.1%   6.4    2s
H 5876  3164                     -188226.7818 -210900.89  12.0%   6.3    3s
  9281  5152 -192568.56   62   50 -188226.78 -209633.03  11.4%   6.2    5s
 24765 12393 -194237.85   42   50 -188226.78 -205750.35  9.31%   6.4   10s
H25267 11765                     -190619.4788 -205716.49  7.92%   6.4   10s
 39314 15223     cutoff   50       -190619.48 -203402.09  6.71%   6.4   15s
 51427 17115 -194173.45   47   25 -190619.48 -201649.93  5.79%   6.4   20s
 67287 17611     cutoff   36       -190619.48 -199597.77  4.71%   6.4   25s
 82144 15576     cutoff   42       -190619.48 -197694.67  3.71%   6.3   30s
...
```

MIP – example 1

- Try changing the focus of the search…
  - `MIPFocus=1`: focus on finding feasible solutions
  - `MIPFocus=2`: focus on proving optimality
  - `MIPFocus=3`: focus on improving the bound

```
     Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
...
     0     0 -209067.20    0   70 -162837.12 -209067.20  28.4%     -    1s
     0     5 -209067.20    0   70 -162837.12 -209067.20  28.4%     -    2s
H   79    66                      -165743.5947 -208071.05  25.5%   3.7    2s
H 1045   871                      -167148.0384 -208070.60  24.5%   5.2    2s
  1238  1014 -193529.37   37   55 -167148.04 -206993.76  23.8%   5.1    5s
H 1647  1140                      -168514.0578 -203113.35  20.5%   6.8    8s
* 1787   994              61      -173659.7799 -203113.35  17.0%   7.0    8s
* 2018   993              54      -175972.1389 -202292.18  15.0%   7.1    8s
H 2692   615                      -187397.9703 -200840.93  7.17%   6.9    8s
  2829   688 -187728.90   32   47 -187397.97 -200681.75  7.09%   7.4   10s
H 4228   578                      -190619.4788 -198075.82  3.91%   7.2   11s
```

MIP – example 2

- MIP log looks like this…

```
     Nodes    |    Current Node    |     Objective Bounds      |     Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 164800.976    0 1096 1169768.62 164800.976  85.9%     -  186s
     0     0 165096.286    0 1607 1169768.62 165096.286  85.9%     -  526s
H    0     0                     481589.35817 165096.286  65.7%     - 1065s
     0     0 165452.400    0 1705 481589.358 165452.400  65.6%     - 1419s
     0     0 165566.561    0 1774 481589.358 165566.561  65.6%     - 1783s
     0     0 165719.831    0 1778 481589.358 165719.831  65.6%     - 2368s
     0     0 165849.075    0 1924 481589.358 165849.075  65.6%     - 2819s
…
```

- Issues:
  - Bound moving very slowly
  - “Stuck” at the root

MIP – example 2

- In extreme cases, try turning off cuts (set `Cuts=0`)…

```
     Nodes    |    Current Node    |     Objective Bounds      |     Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 164800.976    0 1108 1169768.62 164800.976  85.9%     -  225s
     0     0 164800.976    0 1004 1169768.62 164800.976  85.9%     -  249s
H    0     0                     503233.56923 164800.976  67.3%     -  406s
     0     2 164800.976    0  805 503233.569 164800.976  67.3%     -  425s
     2     4 164888.973    2 1009 503233.569 164800.976  67.3%  4039  437s
     3     5 164800.976    2  889 503233.569 164800.976  67.3%  2861  440s
     5     7 164899.272    3  998 503233.569 164800.976  67.3%  2285  454s
     6     7 164998.929    4 1047 503233.569 164800.976  67.3%  2341  465s
     7    10 164837.100    4 1007 503233.569 164800.976  67.3%  2201  481s
     9    10 165308.611    5 1100 503233.569 164837.100  67.2%  2660  497s
    11    16 164837.100    5 1005 503233.569 164837.100  67.2%  2200  509s
    15    18 165265.546    6 1075 503233.569 164837.100  67.2%  2080  531s
    19    23 164837.100    6 1004 503233.569 164837.100  67.2%  1817  557s
    24    29 164864.772    7 1026 503233.569 164864.772  67.2%  1897  571s
H   28    30                     435540.73217 164864.772  62.1%  1722  874s
H   29    32                     371622.48999 164864.772  55.6%  1663  874s
...
```

MIP – example 2

- Change MIP strategies (set `ImproveStartTime=3600`)…

```
…
   836   828 170012.416  113 1107 371622.490 164864.772  55.6%   901 3614s

Resetting heuristic parameters to focus on improving solution
(using Heuristics=0.5 and RINS=10)...

   913   905 170302.813  121 1073 371622.490 164864.772  55.6%   837 4349s
   917   906 170306.908  122 1073 371622.490 164864.772  55.6%   834 5120s
H  920   904                      236882.29084 164864.772  30.4%   833 5120s
H  924   908                      235325.82038 164864.772  29.9%   829 5120s
H  935   917                      235325.82033 164864.772  29.9%   823 6310s
H  941   921                      235325.81953 164864.772  29.9%   820 11402s
   945   930 173125.061  126 1055 235325.820 164864.772  29.9%   819 20667s
   949   936 172549.331  125 1077 235325.820 164864.772  29.9%   818 21413s
H  952   936                      223946.40672 164864.772  26.4%   816 21413s
H  956   938                      223050.75538 164864.772  26.1%   813 21413s
   957   942 172391.221  126 1052 223050.755 164864.772  26.1%   812 29280s
   961   940 173719.366  127 1048 223050.755 164864.772  26.1%   811 29963s
  1001   972 172401.160  130 1050 223050.755 164864.772  26.1%   794 30970s
H 1030   994                      218867.25201 164864.772  24.7%   784 30970s
…
```

MIP – example 3

- Solve progress slows, ‘top’ (or Task Manager) shows Gurobi not making good progress…

```
top - 14:27:11 up 22 days, 22:07,  3 users,  load average: 4.15, 4.05, 3.99
Tasks:  73 total,   1 running,  71 sleeping,   1 stopped,   0 zombie
Cpu(s):  5.9%us,  0.5%sy,  0.0%ni, 32.4%id, 61.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8193828k total,  8144716k used,    49112k free,      284k buffers
Swap: 19800072k total,  3337364k used, 16462708k free,     2108k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3414 rothberg  20   0 10.1g 7.4g 1636 D   23 95.3 657:27.82 gurobi_cl
  207 root      15  -5     0    0    0 S    2  0.0   0:21.37 kswapd0
    1 root      20   0  4020  168  168 S    0  0.0   0:01.22 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.51 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.40 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.13 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.54 migration/1
```

MIP – example 3

- Use node files
  - `NodefileStart` parameter

- Performance penalty typically less than 10%

MIP – example 4

GUROBI OPTIMIZATION

- Progress stalls, not short on memory…

```
 Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 3817190.68    0  544          - 3817190.68     -      -    2s
H    0     0                    1.555212e+09 3817190.68   100%     -    2s
H    0     0                    1.540152e+09 3817190.68   100%     -    2s
H    0     0                    1.496837e+09 3817190.68   100%     -    2s
H    0     0                    1.639804e+08 3817190.68  97.7%     -    3s
...
  5468  4240 3830021.31  435   96 8247474.49 3822445.39  53.7%  36.7   73s
H 5724  3251                    7554727.8619 3822445.39  49.4%  35.7   73s
* 5724  3251             484    7554727.8619 3822445.39  49.4%  35.7   73s
H 6526   921                    3830692.9887 3822445.39  0.22%  33.0   74s
  6526   921             535    3830692.9887 3822445.39  0.22%  33.0   74s
...
 22945  6545 3824602.97  208   45 3824990.61 3824448.25  0.01%  27.5  175s
*23183  6450             348    3824966.4316 3824448.25  0.01%  27.3  175s
 27795 10035 3824931.45  147   52 3824966.43 3824448.56  0.01%  25.6  180s
...
3975393 2423369     cutoff  123      3824938.94 3824541.29  0.01%  23.0 7200s
...
```

MIP – example 4

- Adjust your termination criteria
- Model data often have estimation errors
- Default `MIPGap` (0.01%) is overkill for many models

Additional helpful MIP parameters

- Change branching strategy using `VarBranch`
- Change settings for `Cuts` or `Heuristics`
- If no solution is found after the root node try `PumpPasses, ZeroObjNodes, MinRelNodes`

Parameter pitfalls

- Don’t over-tune parameters
  - Default values are carefully selected, based on thousands of models
  - Avoid setting parameters unless they make a big improvement across multiple test models
  - "Less is more"

- Don’t assume parameters that were effective for another solver are ideal for Gurobi

- If there is a new Gurobi major release try the defaults first

“Gurobi crashed”

- “Crash” means different things to different people

- Most unexpected behavior can be easily diagnosed via simple tests
  - Testing the Status attribute after optimization (optimal, time limit, etc.)
  - Handling exceptions
  - Reviewing the Gurobi logs

- Based on support cases, a bug in Gurobi libraries is very rare

Examples of unexpected behavior

GUROBI OPTIMIZATION

Symptom	Cause
Windows blue screen	Hardware problem – user program cannot crash Windows
Segmentation fault (Linux, Mac)	Problem in your code – pointers in C/C++
Fails to start optimization	Invalid license key
Solver returned no solution	Model is infeasible
Binary variable equals .0000000003	MIP is solved via LP relaxations – check IntFeasTol tolerance value
Violates constraint by .0000000004	LP constraints are satisfied to tolerances – check FeasibilityTol value
Objective is 0.002% worse than mathematical optimum	Termination criterion was reached – check TimeLimit and MIPGap
No log entries for a long time	Large or difficult model
Takes a long time to solve	Large or difficult model

More resources

- Sections in the Reference Manual
  - Logging
  - Parameter Guidelines
  - Parameter Tuning

- Gurobi support: support@gurobi.com