Model troubleshooting & performance tuning

GUROBI
OPTIMIZATION

Runtime issues

- Code fails to run
- Unexpected result
- Slow performance

- We will discuss
  - Specific symptoms of runtime issues
  - What causes them
  - How to troubleshoot them

Symptoms of code failing to run

- Blue screen (Windows)
- Segmentation fault (Linux, Mac)
- Computer is frozen
- No log or progress information
- No solution
- Error messages

Top causes of code failure

- Improperly written code
  - Failing to catch exceptions
  - Failing to test solution status
- License key issue
- Data errors
- Low on physical memory
- Hardware failure
- …
- Gurobi bug

Diagnosing when code fails to run

- What exactly is failing?
- What do you need to do to make it fail?
- Can you reproduce it reliably or only randomly?
- Where in the code does it fail?
  - Model creation
  - Solving
  - Callbacks
  - Retrieving solution values

Troubleshooting code failure situations

- Determine when and where it occurs
- Catch exceptions
- Use system profiling tools
  - Ex: System Monitor, top, Activity Monitor, profiler in compiler, etc.
- Try model file independently from code
- Try different hardware or the cloud

Catching exceptions

- Easy to test for exceptions in OO interfaces:

```
try:
  m = read(sys.argv[1])
  m.optimize()
  for v in m.getVars():
    print v.VarName, v.X
except GurobiError as e:
  print "Error:", e
```

- With C, test the return code for every call to the Gurobi API

- Don't be sloppy – always test for exceptions!
  - Many support requests to Gurobi could be avoided by testing for exceptions and reviewing the exception values

Try model independently from code

- In your code, export the model as MPS file:

```
m.update();
m.write("test.mps");
```

  - Use MPS files, not LP files
    - Note: constant offset (ObjCon attribute) is *not* saved in MPS file

- Test model file separately:
  - Command-line utility

```
gurobi_cl test.mps
```

  - Python shell

```
m=read("test.mps")
m.optimize()
```

Saving full state

- Use files to write and read parameters, warm starts
  - Non-default parameters: .PRM file
  - MIP start values: .MST file
  - Initial LP basis: .BAS file

- Export in your code:

```
m.update();
m.write("test.mps");
m.write("test.prm");
m.write("test.mst");
```

- Test model file separately:
  - Command-line utility

```
gurobi_cl inputfile=test.prm inputfile=test.mst test.mps
```

  - Python shell

```
m=read("test.mps")
m.read("test.prm")
m.read("test.mst")
m.optimize()
```

Symptoms of unexpected results

- Model is reported as infeasible
- A known solution is not feasible
- Solution appears to be suboptimal
- Solution violates some known constraints

Causes of unexpected results

- Modeling errors
- Unexpected data
- Numerical issues

Troubleshooting unexpected results

- Review model statistics
- Inspect LP file
- Test known solution
- Try infeasibility detection

Model statistics

- Does the model match your expectations for size, coefficients?

- Model.printStats() function in Python API gives a model overview

```
m=read('misc07')
m.printStats()
```

```
Statistics for model MISC07 :
  Linear constraint matrix    : 212 Constrs, 260 Vars, 8619 NZs
  Variable types              : 1 Continuous, 259 Integer
  Matrix coefficient range    : [ 1, 700 ]
  Objective coefficient range : [ 1, 1 ]
  Variable bound range        : [ 0, 1 ]
  RHS coefficient range       : [ 1, 300 ]
```

- These stats are printed in the log, and they are available via the attribute interface

LP file

- LP file uses algebraic syntax
- With meaningful variable and constraint names, LP files can be a valuable debugging tool

```
Maximize
  x + y + 2 z
Subject To
 c0: x + 2 y + 3 z <= 4
 c1: x + y >= 1
Bounds
Binaries
 x y z
End
```

Testing a known solution

MIP optimality

- Use solution as start values, then search for improvements
  - Use MIP start file (.MST) or Start variable attribute

- Python example

```
m=read("test.mps")
m.read("test.mst")
m.optimize()
```

Feasibility

- Set bounds to known solution
  - No automatic feature but easy to do with APIs

- Python example

```
m=read("test.mps")
x=m.getVarByName("x")
x.setAttr("LB", 1.0)
x.setAttr("UB", 1.0)
```

Infeasibility detection tools

IIS

- Finds a minimal set of constraints that conflict
- Attributes indicate what cannot be satisfied
- Useful for model development

FeasRelax

- Finds a solution with the minimum constraint violation
- Retrieve added artificial slack variables to find violations
- Useful for deployed application

Symptoms of slow performance

- It's slow!

Causes of slow performance

- Numerical issues
- Memory limits
- 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
- Parameter tuning tool
- Performance guidelines
- Send test models 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?
- Is it comparably slow on different computers?
  - Gurobi can provide temporary licenses for testing on other computers
  - Try the 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

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
- Best to 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
```

Performance guidelines by category

- General issues
- Continuous optimization
- Integer optimization

- Let's examine these in detail

Performance guidelines by category

- General issues
- Continuous optimization
- Integer optimization

General issues

- Model initialization
- Presolve
- Numerical issues
- Memory

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
- With Gurobi OO interfaces, take advantage of lazy updates
  - Only call update function when necessary to reference new objects

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

Numerical issues

- Wide range of model coefficients can ruin performance
  - Can affect continuous or integer models
- Look for: Unusual messages in solution log
  - Ex: Markowitz tolerance increased

Memory

- Insufficient memory can wreck 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 use disk to store MIP node info
- Memory is cheap; no need to skimp

Sources of bottlenecks

- General issues
- Continuous optimization
- Integer optimization

Continuous algorithms

- Dual simplex
- Primal simplex
- Barrier

- Concurrent (LP)
  - Use multiple algorithms at the same time on multiple processor cores
  - Multiple algorithms makes it very robust
  - Requires more memory

Concurrent

- Two concurrent modes
  - Fast
    - First algorithm that finishes wins
    - Occasionally, there is a near-tie
      - In this case, the solution can vary on multiple runs
  - Deterministic
    - Consistent, repeatable winner declaration
    - Usually much slower than fast mode
      - Winning method determined by internal metrics, not runtime

LP Performance

- Mean runtime ratios (quad-core Xeon E3-1240 v3):
  - ~210 models with runtime > 1s
  - Concurrent: dual on 1 core, barrier on 3

	Geometric mean
Dual simplex	1.00
Primal simplex	2.11
Barrier	0.51
Concurrent	0.40
Deterministic concurrent	0.44

QP Performance

- Mean runtime ratios (quad-core i7-2600):
  - 14 models with runtime > 1s

	Geometric mean
Dual simplex	1.00
Barrier	0.02

Typical defaults for continuous optimization

- LP Concurrent
- QP Barrier
- MIP root Dual simplex or 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

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 concurrent
- Different optimal basis possible when dual and barrier runtimes are very close

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

LP – Example 2

- Solve model pds-100 on a 4-core i7-3770K
- Dual…

```
Solved in 112881 iterations and 15.36 seconds
```

- Barrier…

```
Solved in 78612 iterations and 40.41 seconds
```

- Concurrent…

```
Solved with dual simplex
Solved in 112881 iterations and 21.03 seconds
```

- "I thought you said that concurrent was the fastest option"

The Fastest Solver

- 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 3

- Small LP solves quickly
- Larger LP solves exponentially slower

- "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
- Crossover
  - Select strategy used for transforming barrier solution to basic solution
  - No effect for QP or QCP models
- CrossoverBasis
  - Select strategy used to construct initial basis from barrier solution

Sources of bottlenecks

- General issues
- Continuous optimization
- Integer optimization

ID why your MIP is 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
 96108 11245     cutoff   34       -190619.48 -195664.78  2.65%   6.3   35s
 109391 4347 -192569.29   47   30 -190619.48 -193256.98  1.38%   6.1   40s
...
```

- What should you try?

MIP – Example 1

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

```
    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
- What should you try?

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
```

- What should you try?

MIP – Example 3

- Use node files
  - NodefileStart parameter

- Performance penalty typically less than 10%

MIP – Example 4

- 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
...
```

- What should you try?

MIP – Example 4

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

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

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

More resources

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

- Gurobi support: support@gurobi.com