Performance variability in mixed integer programming

Emilie Danna
ILOG CPLEX R&D
edanna@ilog.com

MIP 2008
Example: 10 teams, CPLEX 11, Linux

Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time = 0.01 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.05 sec.

<table>
<thead>
<tr>
<th>Node</th>
<th>Left</th>
<th>Objective</th>
<th>Infs</th>
<th>Best Integer</th>
<th>Best Node</th>
<th>ItCnt</th>
<th>Gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>917.0000</td>
<td>140</td>
<td></td>
<td>917.0000</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>165</td>
<td></td>
<td>924.0000</td>
<td>1969</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>167</td>
<td></td>
<td>924.0000</td>
<td>2348</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>924.0000</td>
<td>175</td>
<td></td>
<td>924.0000</td>
<td>2731</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
<td>0</td>
<td>924.0000</td>
<td></td>
<td>924.0000</td>
<td>2731</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Clique cuts applied: 16
Zero-half cuts applied: 3
Gomory fractional cuts applied: 1

Solution pool: 1 solution saved.

MIP - Integer optimal solution: Objective = 9.2400000000e+02
Solution time = 0.41 sec. Iterations = 2731 Nodes = 0
Example: 10 teams, CPLEX 11, AIX

Tried aggregator 1 time.
MIP Presolve eliminated 20 rows and 425 columns.
Reduced MIP has 210 rows, 1600 columns, and 9600 nonzeros.
Presolve time = 0.00 sec.
Clique table members: 170.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.18 sec.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Left</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>100+</td>
</tr>
<tr>
<td>1000</td>
<td>520</td>
</tr>
<tr>
<td>*</td>
<td>1425</td>
</tr>
</tbody>
</table>

Clique cuts applied: 12
Zero-half cuts applied: 4
Gomory fractional cuts applied: 2

Solution pool: 2 solutions saved.

MIP - Integer optimal solution: Objective = 9.2400000000e+02
Solution time = 41.39 sec. Iterations = 122948 Nodes = 1426
Definition (1)

- Performance variability
  - change in performance (solving time, # nodes, # iterations, …)
  - for the same model
  - created by a change in the solver or the environment
  - that is seemingly performance neutral

- in short, change in performance we do not understand
The topic of this talk is not (but is related to)

- Change in performance due to improving algorithms
- Change in performance due to change in MIP formulation for the same problem
- Change in performance due to change in data
Questions

- What is the **extent** of variability?
- What are the **causes** of variability?
  - Variability generator
    - How to generate variability in order to study it?
  - Solver
    - What algorithms or implementations cause variability?
    - Can we make solvers more robust and therefore faster?
  - Model
    - What characteristics make a model variable?
- What are the **consequences** of variability?
Experimental setup (1)

- 368 models that solve to 0.01% gap between 10 and 100 seconds with CPLEX 11
  - Homogeneous set
  - Large enough for statistical analysis

- Performance is measured as the number of simplex iterations
  - We are really interested in the solving time
  - But solving time is difficult to measure precisely
  - The number of iterations is a better proxy than the number of nodes
For each model

- Performance with the original conditions
  - \text{niter}(\text{orig})

- Performance with perturbed conditions
  - for $i = 1 \ldots 10$ different instances of the same variability generator $g$
  - \text{niter}(g,i)

- Variability ($g$) is \textit{estimated} as
  - standard deviation of the sample
  - \frac{\text{niter}(g,i)}{\text{niter}(\text{orig})}, i = 1 \ldots 10
First variability generator: permutations
Definition and properties

- Random permutations of rows and columns
- Should be performance neutral
- Affects all components of the solver
Distribution of variability (1)
In practice, what does a variability of ‘v’ mean?

- **Variability ~ 0.1**: small and reasonable variations

<table>
<thead>
<tr>
<th>Variability</th>
<th>0.1053</th>
<th>0.89</th>
<th>1.08</th>
<th>0.86</th>
<th>1.02</th>
<th>0.89</th>
<th>0.88</th>
<th>0.87</th>
<th>0.98</th>
<th>1.09</th>
<th>1.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variability</td>
<td>0.0900</td>
<td>1.06</td>
<td>1.08</td>
<td>0.97</td>
<td>1.21</td>
<td>1.03</td>
<td>0.98</td>
<td>0.91</td>
<td>1.1</td>
<td>1.1</td>
<td>1.17</td>
</tr>
<tr>
<td>Variability</td>
<td>0.1183</td>
<td>0.96</td>
<td>1.18</td>
<td>0.88</td>
<td>0.85</td>
<td>0.88</td>
<td>0.96</td>
<td>0.96</td>
<td>1.09</td>
<td>0.82</td>
<td>1.08</td>
</tr>
</tbody>
</table>

- **Variability ~ 1**: already quite large

<table>
<thead>
<tr>
<th>Variability</th>
<th>0.9728</th>
<th>0.69</th>
<th>0.9</th>
<th>2.46</th>
<th>0.67</th>
<th>1.03</th>
<th>0.95</th>
<th>0.9</th>
<th>1.1</th>
<th>2.49</th>
<th>3.47</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variability</td>
<td>0.9448</td>
<td>1.7</td>
<td>0.4</td>
<td>2.39</td>
<td>0.9</td>
<td>0.74</td>
<td>0.63</td>
<td>0.35</td>
<td>0.51</td>
<td>3.14</td>
<td>0.81</td>
</tr>
</tbody>
</table>

- **Variability > 10**: terrible

<table>
<thead>
<tr>
<th>Variability</th>
<th>10.4249</th>
<th>3.95</th>
<th>23.9</th>
<th>5.11</th>
<th>4.23</th>
<th>22.16</th>
<th>8.45</th>
<th>1.46</th>
<th>30.07</th>
<th>5.09</th>
<th>3.17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variability</td>
<td>18.9003</td>
<td>3.78</td>
<td>2.1</td>
<td>2.35</td>
<td>1.92</td>
<td>3.02</td>
<td>3.72</td>
<td>62.29</td>
<td>1.41</td>
<td>3.5</td>
<td>1.48</td>
</tr>
<tr>
<td>Variability</td>
<td>88.7852</td>
<td>1.12</td>
<td>229.4</td>
<td>0.92</td>
<td>0.84</td>
<td>0.65</td>
<td>0.67</td>
<td>0.73</td>
<td>0.65</td>
<td>0.59</td>
<td>191.1</td>
</tr>
</tbody>
</table>
Distribution of variability (2)

- **Models for which v > 10**
  - The number computed is a poor estimation of the true variability for these models
  - In the following, we truncate v > 10 to v = 10 to have a more “robust” estimate

<table>
<thead>
<tr>
<th>Median (v)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>v &lt;= 0.1</td>
<td>16.6%</td>
</tr>
<tr>
<td>0.1 &lt; v &lt;= 1</td>
<td>64.4%</td>
</tr>
<tr>
<td>1 &lt; v &lt;= 10</td>
<td>15.8%</td>
</tr>
<tr>
<td>v &gt; 10</td>
<td>3.3%</td>
</tr>
<tr>
<td>Median (v)</td>
<td>0.29</td>
</tr>
</tbody>
</table>
Distribution of variability (3)
Are permutations really performance neutral?

- Performance on permuted vs. original models
  - +30% more time
  - +8% more branch & bound nodes
  - +13% more simplex iterations

- Why?
Cache misses when solving the root LP on permuted vs. original models (measured with cachegrind):

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Geometric mean</th>
<th>Impact of one cache miss</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 data cache</td>
<td>+34%</td>
<td>+36%</td>
<td>~10 cycles</td>
</tr>
<tr>
<td>L2 data cache</td>
<td>+4%</td>
<td>+13%</td>
<td>~200 cycles</td>
</tr>
</tbody>
</table>

The increase in cache misses is a good candidate to explain the increase in time per iteration and time per node.

We presume the locality also affects the discrete components of the MIP solver.
Quantifying locality: matrix dispersion

\[ A = (a_{i,j})_{i=1..m, j=1..n} \]

\[ nz_j = \# \{ i \in 1..m, a_{i,j} \neq 0 \} \]

\[ \delta_{i,j,j+1} = 1 \text{ if } a_{i,j} = 0 \text{ and } a_{i,j+1} \neq 0 \quad \text{or} \quad a_{i,j} \neq 0 \text{ and } a_{i,j+1} = 0 \]

\[ \delta_{i,j,j+1} = 0 \text{ otherwise} \]

\[ Dispersion(A) = \frac{\sum_{j=1}^{n-1} \sum_{i=1}^{m} \delta_{i,j,j+1}}{2 \sum_{j=1}^{n} nz_j} \]

\[ 0 \leq Dispersion(A) \leq 1 \]
Matrix dispersion and permutations

- +91% dispersion on permuted models compared to original models
  - The order of rows and columns chosen by a human modeler creates matrices with a small dispersion

- But no correlation between the increase in dispersion and the performance degradation
Permutations: SCIP

- SCIP with CPLEX as the LP solver
- Subset of 135 models: models that solve to 0.01% gap with SCIP in less than 500 seconds
Permutations: SCIP vs. CPLEX

<table>
<thead>
<tr>
<th>Variability Range</th>
<th>CPLEX</th>
<th>SCIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median (v)</td>
<td>0.29</td>
<td>0.24</td>
</tr>
<tr>
<td>v &lt;= 0.1</td>
<td>8.1%</td>
<td>19.3%</td>
</tr>
<tr>
<td>0.1 &lt; v &lt;= 1</td>
<td>76.3%</td>
<td>66.7%</td>
</tr>
<tr>
<td>1 &lt; v &lt;= 10</td>
<td>13.3%</td>
<td>12.6%</td>
</tr>
<tr>
<td>v &gt; 10</td>
<td>2.2%</td>
<td>1.5%</td>
</tr>
</tbody>
</table>

- Variability and performance need to be interpreted together
  - SCIP is about 4 times slower than CPLEX on this subset of 135 models (and about 20 times slower on the entire set of 368 models)
  - No correlation between variability for CPLEX and SCIP
Second variability generator: random generator initialization
Definition and properties

- Change the seed of the random number generator
- Should really be performance neutral
- Affects mainly heuristics
  - But once the path is changed, everything is affected
- Experiments with CPLEX on Linux and AIX
Comparison of variabilities

Correlation = 0.7 (0.59 without truncation)
Difference between AIX and Linux?

- Correlation = 0.51 (0.04 without truncation)
Third variability generator: degenerate pivots
Definition and properties

- Make a few random dual degenerate pivots at the root
- The root LP basis is different, therefore all LP-based components that follow give different results: cuts, branching, most heuristics, …
- Cannot be applied to some problems
  - Either there are no degenerate pivots
  - Or CPLEX refuses to do those pivots for numerical reasons
Comparison of variabilities

- Correlation = 0.53 (0.93 without truncation)
### Conclusions on variability generators (1)

- No clear winner: each generator has its advantages and drawbacks

<table>
<thead>
<tr>
<th>Generator / property</th>
<th>Permutations</th>
<th>Seed for random generator</th>
<th>Degenerate pivots</th>
<th>Change of platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance neutral</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Many instances of the generator can be applied</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes / No</td>
<td>No</td>
</tr>
<tr>
<td>Applies to all types of problems</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Affects all components of the solver</td>
<td>Yes</td>
<td>No</td>
<td>Almost</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Conclusions on variability generators (2)

- Variability depends on the model and the solver
- Given a model and a solver, variability does not depend much on the generator
Causes of variability
Possible causes

- **Component of the solver**
  - Heuristics, branching, cuts, …

- **Aspect of optimization**
  - Is obtaining the optimal solution less or more robust than proving its optimality?

- **Characteristic of the model**
  - Landscape of optimal solutions, numerical instability, …
Variability when the optimal solution is known
Variability when the optimal solution is known

<table>
<thead>
<tr>
<th>Median (v)</th>
<th>CPLEX without MIP start</th>
<th>CPLEX with the optimal MIP start</th>
</tr>
</thead>
<tbody>
<tr>
<td>v &lt;= 0.1</td>
<td>51.9%</td>
<td>65.8%</td>
</tr>
<tr>
<td>0.1 &lt; v &lt;= 1</td>
<td>39.4%</td>
<td>29.1%</td>
</tr>
<tr>
<td>1 &lt; v &lt;= 10</td>
<td>6.2%</td>
<td>4.6%</td>
</tr>
<tr>
<td>v &gt; 10</td>
<td>2.4%</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

- Finding the optimal solution is a significant cause of variability, especially for extreme cases.
Variability and number of alternative optima

- No correlation
Variability and numerical instability

- No correlation
Conclusions on causes of variability

- One known factor: finding the optimal solution
- Many unknown causes
- Future work: looking at the correlation between variability and a given model characteristic is simplistic
  - The correlation is very much influenced by extreme cases
  - Variability might be determined by several factors at once
  - Looking in more details at the results (did not find the essential cut, did not find the optimal solution, did not branch on the right variable, …?) should give more insight
Consequences of variability
Consequences for benchmarking (1)

- The performance difference between code A and code B measures
  - The true effect of the code change
  - Noise (variability)

- The analysis of benchmarking results needs to distinguish between the two
Consequences for benchmarking (2)

- **Large model sets**
  - Or, artificially increase the size of test sets with a variability generator

- **Statistical analysis of results**
  - The majority of papers use descriptive statistics
  - Performance profiles are a step in the right direction
  - But benchmarking observations provide only an estimation of the true effect
  - Therefore, we need inferential statistics (statistical tests, confidence intervals, …) to answer questions such as:
  - How likely is it that the performance difference observed is created by variability rather than by my algorithmic change?
Consequences for R&D

- Variability is annoying

- But it is an opportunity for
  - performance improvement
  - better understanding what makes optimization hard in practice