Optimization Direct, CPLEX and Very Large Optimization Models

Alkis Vazacopoulos
Robert Ashford
Optimization Direct
Summary

• Introduce Optimization Direct Inc.
• Explain what we do
• Look at getting the most from optimization
  • some modeling issues
  • optimizer issues
• Example of large scale optimization: scheduling heuristic
Optimization Direct

- IBM Business Partner
- Sell CPLEX optimization Studio
- More than 30 years of experience in developing and selling Optimization software
- Experience in implementing optimization technology in all the verticals
- Sold to end users – Fortune 500 companies
- Train our customers to get the maximum out of the IBM software
- Help the customers get a flying start and get the most from optimization and the software right away
Background

• Robert Ashford
  • Robert co-founded Dash Optimization in 1984. Helped pioneer the development of new modeling and solution technologies – the first integrated development environment for optimization – in the forefront of technology development driving the size, complexity and scope of applications. Dash was sold in 2008 and Robert continued leading development within Fair Isaac until the Fall of 2010, Dr. Ashford subsequently, co-founded Optimization Direct in 2014.

• Alkis Vazacopoulos
  • Alkis is a Business Analytics and Optimization expert. From January 2008 to January 2011 he was Vice President at FICO Research. Prior to that he was the President at Dash Optimization, Inc. where he worked closely with end users, consulting companies, OEMs/ISVs in developing optimization solutions.
Get more from Optimization

- **Modeling**
  - Use modeling language and ‘Developer Studio’ environment
    - Such as OPL and CPLEX Optimization Studio
    - Easier to build, debug, manage models
  - Exploit (data) sparsity
  - Keep formulations tight

- **Optimization**
  - Many models solve out-of-the-box
  - Others (usually large) models do not
    - Tune optimizer
    - Distributed MIP
    - Use Heuristics
Modeling: Use Sparsity

Avoid unnecessary looping: example of network model

\[ X_{sdij} = \text{traffic on link (i,j) from route (s,d)} \]

\[ \sum_j X_{sdlj} - \sum_i X_{sdil} = 0, \text{ for all } s,d,l \in \{\text{Nodes}\} \]

Coded as:

```plaintext
int NbNodes = ...;
range Nodes = 1..NbNodes;
dvar float+ X[Nodes][Nodes][Nodes][Nodes][Nodes] = ...;

forall(s in Nodes, d in Nodes, l in Nodes : s != d && s != l && d != l)
    XFlow:
        sum(j in Nodes) X[s][d][l][j] - sum(i in Nodes) X[s][d][i][l][l] == 0;
```

is inefficient: loops over all combinations
Modeling: Use Sparsity

More efficient: only loop over existent links and routes

\( X_{kl} = \text{traffic on link } l \text{ from route } k \)

\[ \sum_{j: (n,j) \in \{\text{links}\}} X_{k(n,j)} - \sum_{i: (i,n) \in \{\text{links}\}} X_{k(l,n)} = 0, \text{ all } k \in \{\text{Links}\}, n \in \{\text{Nodes}\} \]

coded as

```c
int NbNodes = ...;
range Nodes = 1..NbNodes;
tuple link { int s; int d;}
{link} Links = ...;
dvar float+ X[Links][Links];

forall( k in Links, n in Nodes : n != k.s && n != k.d )
  XFlow:
    sum(<n,j> in Links) X[k][<n,j>] - sum(<i,n> in Links) X[k][<i,n>] == 0;
```
Tight Formulations

• Make LP feasible region as small as possible whilst containing integer feasible points

• ‘big-M’/indicator constraints well known:
  
  \[ y - Mx \leq 0, \quad y \geq 0, \quad s \text{ binary. Identify directly to solver.} \]

• Take care with others. Customer example: \( x_i \) binary

  \[ 2x_1 + 2x_2 + x_3 + x_4 \leq 3 \] is worse than

  \[ x_1 + x_2 \leq 1 \] and \( x_1 + x_2 + x_3 + x_4 \leq 2 \)

  \[ 2x_1 + 2x_2 + x_3 + x_4 \leq 2 \] is worse than

  \[ x_1 + x_2 \leq 1 \] and \( x_1 + x_2 + x_3 + x_4 \leq 1 \)
Driving the Optimizer

- Performance of modern commercial software and hardware in a different league from ten years ago
  - Hardware
    - Around 10 X faster than 10 years ago
    - Depends on characteristic of interest
  - Software
    - CPLEX leads
    - 100X ~ 1000X faster than best open source
    - Tunable to specific model classes
- Exploits multiple processor cores
- Exploits multiple machine clusters
<table>
<thead>
<tr>
<th>Hardware characteristics</th>
<th>1990</th>
<th>2005</th>
<th>2014</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM PS/2 80-111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor speed</td>
<td>20 MHz</td>
<td>3.2 GHz</td>
<td>4.4 GHz</td>
<td>220</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Time for a FP multiply</td>
<td>1.4 – 2.8 ms.</td>
<td>0.08† – 0.3 ns.</td>
<td>0.008† – 0.2 ns.</td>
<td>7,000 – 350,000†</td>
</tr>
<tr>
<td>Typical memory size</td>
<td>4 Mbyte</td>
<td>2 GByte</td>
<td>16 GByte</td>
<td>4000</td>
</tr>
<tr>
<td>Memory speed</td>
<td>85 ns.</td>
<td>45 ns.</td>
<td>12 ns</td>
<td>7</td>
</tr>
<tr>
<td>L2 cache size</td>
<td>None</td>
<td>1 Mbyte</td>
<td>8 Mbyte</td>
<td></td>
</tr>
<tr>
<td>Typical disk capacity</td>
<td>115 Mbyte</td>
<td>120 GByte</td>
<td>1 TByte</td>
<td>9,000</td>
</tr>
</tbody>
</table>

† using the vector facility.
Where’s the Problem?

• Many models now solved routinely which would have been impossible ('unsolvable') a few years ago

• BUT: have super-linear growth of solving effort as model size/complexity increases

• AND: customer models keep getting larger
  • Globalized business has larger and more complex supply chain
  • Optimization expanding into new areas, especially scheduling
  • Detailed models easier to sell to management and end-users
Getting more difficult

- Solver has to
  - (Presolve and) solve LP relaxation
  - Find and apply cuts
  - Branch on remaining infeasibilities (and find and apply cuts too)
  - Look for feasible solutions with heuristics all the while

- Simplex relaxation solves theoretically NP, but in practice effort increases between linearly and quadratic

- Barrier solver effort grows more slowly, but:
  - cross-over still grows quickly
  - usually get more integer infeasibilities
  - can’t use last solution/basis to accelerate

- Cutting grows faster than quadratic: each cuts requires more effort, more cuts/round, more rounds of cuts, each round harder to apply.

- Branching is exponential: $2^n$ in number of (say) binaries n
What can we do? Tuning the Optimizer

- Models usually solved many times
  - Repeat planning or scheduling process
  - Solve multiple scenarios

- Good algorithms for dynamic parameter/strategy choice

- Expert analysis may do even better trading off solution time and optimality level:
  - LP algorithm choice
  - Kind of cuts and their intensity
  - Heuristic choice, effort and frequency
  - Branching strategies and priorities

- Use more parallelism: more cores, distributed computers

- Sophisticated auto tuner for small/medium sized models
Model/Application Specific Solution Techniques

- Proof of optimality (say to 1%) may be impractical
- Want good solutions to (say) 20%
- Solve smaller model(s)
  - Heuristic approach used e.g. by RINS and local branching
- Use knowledge of model structure to break it down into sub-models and combine solutions
- Prove solution quality by a very aggressive root solve of whole model
Example: Large Scale Scheduling model

- Schedule 314 entities over 64 periods
- Have target number of active periods for each entity
- Entities have mutual interactions within each period
- 94K variables, 213K constraints, 93K binaries
- No usable (say within 30% gap) solution after 2 days run time on fastest hardware (Intel i7 4790K ‘Devil’s Canyon’)
Heuristic Solution Approach

- Solves sequence of sub-models
- Delivers usable solutions (12%-16% gap)
- Takes 4-12 hours run time
- Multiple instances can be run concurrently with different seeds
- Can run on only 2 cores
- Can interrupt at any point and take best solution so far
## Scheduling Model Heuristic Results

<table>
<thead>
<tr>
<th>Seed</th>
<th>Solution</th>
<th>Gap</th>
<th>Total Time</th>
<th>CPLEX Time</th>
<th>CPLEX Its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>119</td>
<td>16%</td>
<td>357</td>
<td>354</td>
<td>6,801,292</td>
</tr>
<tr>
<td>5678</td>
<td>116</td>
<td>14%</td>
<td>277</td>
<td>272</td>
<td>5,427,272</td>
</tr>
<tr>
<td>9012</td>
<td>115</td>
<td>13%</td>
<td>701</td>
<td>698</td>
<td>13,183,197</td>
</tr>
<tr>
<td>21098</td>
<td>118</td>
<td>15%</td>
<td>347</td>
<td>343</td>
<td>7,118,413</td>
</tr>
<tr>
<td>14702</td>
<td>117</td>
<td>15%</td>
<td>230</td>
<td>228</td>
<td>5,960,748</td>
</tr>
<tr>
<td>17083</td>
<td>118</td>
<td>15%</td>
<td>240</td>
<td>237</td>
<td>5,981,049</td>
</tr>
<tr>
<td>31117</td>
<td>118</td>
<td>15%</td>
<td>455</td>
<td>451</td>
<td>7,264,158</td>
</tr>
<tr>
<td>23715</td>
<td>115</td>
<td>13%</td>
<td>262</td>
<td>259</td>
<td>5,959,212</td>
</tr>
<tr>
<td>7039</td>
<td>118</td>
<td>15%</td>
<td>354</td>
<td>351</td>
<td>8,452,977</td>
</tr>
<tr>
<td>27332</td>
<td>123</td>
<td>19%</td>
<td>289</td>
<td>287</td>
<td>5,407,432</td>
</tr>
<tr>
<td>4214</td>
<td>117</td>
<td>15%</td>
<td>403</td>
<td>399</td>
<td>6,746,800</td>
</tr>
<tr>
<td>15375</td>
<td>115</td>
<td>13%</td>
<td>333</td>
<td>328</td>
<td>7,028,708</td>
</tr>
</tbody>
</table>

Best bound of 100 established by separate CPLEX run
Times are in minutes on Intel i7-4790K @ 4.4GHz
Parallel Heuristic Approach

• Run several instances with different seeds simultaneously

• CPLEX callable library very flexible, so
  • Exchange solution information between runs
  • Kill sub-model solves when done better elsewhere

• Improves sub-model selection

• 4 instances run on 4 core i7-4790K
  • Each instance slower than serial case which mostly used 2 cores each
  • Outweighed by information exchange

• Could easily implement on clusters of computers
Parallel Heuristic Behavior
Parallel Heuristic Behavior: 2 cores/instance

Solutions vs Time in Minutes

- 2 Cores 4 Threads Ea
- Serial
Thanks for listening

Robert Ashford
rwa@optimizationdirect.com
www.optimizationdirect.com