IBM ILOG CPLEX
What is inside of the box?

Ricardo Lima
rlima@andrew.cmu.edu

EWO Seminar
Carnegie Mellon University
1. Introduction
   • What is CPLEX? Types of problems. History.
2. Algorithms
   • Optimizers available. Heuristic based algorithms.
3. Parallelization
4. Tools
5. Final remarks
Introduction

CPLEX
Optimization software package
Commercialized by IBM ILOG
Types of problems CPLEX can solve

Mathematical programming problems:

• Linear programming
• Mixed integer programming
• Quadratic programs
• Mixed integer quadratic programs
• Quadratic constrained programs
• Mixed integer quadratic constrained programs

• It is used to solve other problems: MINLP
Maximize: $c^T x$ \hspace{5cm} \text{Objective function}

Subject to: $Ax \leq b$ \hspace{2.5cm} \text{Constraints}
$x \in \mathbb{R}^n$ \hspace{4cm} \text{Decision variables}
$A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m$
Mixed integer linear programming

Maximize: \[ c^T x + d^T y \]
Subject to: \[ Ax + By \leq b \]
\[ x \in \mathbb{R}^n \]
\[ y \in \mathbb{Z}^k_+ \]
\[ A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times k}, c \in \mathbb{R}^n, d \in \mathbb{R}^k, b \in \mathbb{R}^m \]

Integer variables
Maximize: \[ c^T x + \frac{1}{2} x^T Q x \] 
Subject to: \[ Ax \leq b \] 
\[ x \in \mathbb{R}^n \] 
\[ A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m \] 
\[ Q \in \mathbb{R}^{n \times n} \] 

**Remark:** If matrix $Q$ is positive semi-definite then the problem $QP$ is convex.
Quadratic programs

Maximize: \( c^T x + \frac{1}{2} x^T Q x \)
Subject to: \( Ax \leq b \)
\[ x_l \in \mathbb{Z}_+, \ l \in N_l \]
\[ x_j \in \mathbb{R}, \ j \in N_j \]
\[ A \in \mathbb{R}^{m \times n}, \ c \in \mathbb{R}^n, \ b \in \mathbb{R}^m \]
\[ Q \in \mathbb{R}^{n \times n} \]

Remark: If matrix Q is positive semi-definite then the problem QP is convex.
Quadratic constrained programming

Maximize: \[ c^T x + \frac{1}{2} x^T Q x \]
Subject to: \[ Ax \leq b \]
\[ \frac{1}{2} x^T B_i x + a_i x \leq b_i, \ i = 1, \ldots, m_1 \]
\[ x \in \mathbb{R}^n \]
\[ A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m \]
\[ B_i \in \mathbb{R}^{m \times n}, \ i = 1, \ldots, m_1 \]
Quadratic constrained programming

Maximize: \[ c^T x + \frac{1}{2} x^T Q x \]
Subject to: \[ Ax \leq b \]
\[ \frac{1}{2} x^T B_i x + a_i x \leq b_i, \quad i = 1, \ldots, m_1 \]
\[ x_l \in \mathbb{Z}_+, \quad l \in N_l \]
\[ x_j \in \mathbb{R}, \quad j \in N_j \]
\[ A \in \mathbb{R}^{m \times n}, \quad c \in \mathbb{R}^n, \quad b \in \mathbb{R}^m \]
\[ B_i \in \mathbb{R}^{m \times n}, \quad i = 1, \ldots, m_1 \]
Robert Bixby

- Wrote a LP code
- CPplex Optimization Inc.

1988
- CPlex 1.0
- LP optimizer

1992
- CPlex 2.0
- MIP optimizer

1997
- Acquired by ILOG

2009
- Acquired by IBM

2010
- Latest stable release: 12.2

2008 – Bixby, Gu, and Rothberg left ILOG and found Gurobi Optimization.
CPLEX releases history

- **CPLEX 1.0** (1988)
  - LP solver

- **CPLEX 2.0** (1992)
  - Simple B&B
  - Limited cuts

- **CPLEX 6.0** (1998)
  - Simple B&B
  - Limited cuts
  - Simple heuristic
  - Faster dual simplex
CPLEX releases history (cont.)

**CPLEX6.5**
1999
- 5 different node heuristics
- 6 types of cutting planes
  - Knapsack covers
  - GUB covers
  - Flow covers
  - Cliques
  - Implied bounds
  - Gomory mixed integer cuts

**CPLEX 7.0**
2000
- Semi-Continuous and Semi-Integer Variables
- Default LP method: dual simplex.
- Preprocessing
- Cuts:
  - mixed integer rounding
  - disjunctive
  - flow path

**CPLEX 8.0**
2002
- New Methods for Solving LP Models: Sifting
- Concurrent optimization:
  1) Dual Simplex; 2) Barrier method, 3) Primal Simplex, 4) Barrier method
- New QP Capabilities
- 9 types of cutting planes
CPLEX release history (cont.)

CPLEX 9.0 2003
- QCP
- Relaxation Induced Neighborhood Search (RINS)

CPLEX 10.0 2006
- Improvements for MIQPs
- Changes in MIP start behavior
- Feasible Relaxation
- Indicators
- Solution Polishing

CPLEX 11.0 2007
- The solution pool
- Tuning tool
- Parallel mode

CPLEX 12.2 2010
- MIP is faster
- Multi-commodity flow cuts
- Enhanced heuristics
- Enhanced dynamic search
Computational performance

The actual computational performance is the result of a combination of different types of improvements:

- **LP solvers**
  - Pre-processing
  - Algebra for sparse systems
  - Methods: primal, dual, barrier
  - Techniques to avoid degeneracy and numerical difficulties

- **Cutting planes**
  - From theory to practice

- **Heuristics**
  - Node heuristics
  - RINS
  - Polishing

- **Parallelization**
  - Search in B&B
  - Barrier method

Plus the machine improvements
Computational evolution for LPs

In the beginning

• 1952 - (E48, V71) solved in 18 hours, 71 Simplex iterations.
  Orden (1952), Hoffman et al. (1953)

• 1963 - (E99, V77) estimated 120 man days.
  Stigler’s (1945) diet problem

• 1990 - (E26, V71) solved in 8 hours.
  Orchard-Hays (1990)

Evolution reported by Bixby for solving LP problems (1984:2004):

• Algorithms: Primal vs best of Primal/Dual/Barrier 3300x
• Machines: (workstations -> PCs): 1600x
• Net: algorithm x machine 5 300 000x
  5 days/5 300 000 = 0.08 seconds

LP performance

- Computational experiments:

**Size of the LP model:**
- # Equations: 60,390
- # Variables: 69,582

No advanced basis was used

<table>
<thead>
<tr>
<th></th>
<th>CPLEX version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.1</td>
</tr>
<tr>
<td>Primal Simplex</td>
<td>205</td>
</tr>
<tr>
<td>Dual Simplex</td>
<td>281</td>
</tr>
<tr>
<td>Network Simplex</td>
<td>174</td>
</tr>
<tr>
<td>Barrier</td>
<td>97</td>
</tr>
<tr>
<td>Sifting</td>
<td>-</td>
</tr>
</tbody>
</table>
Optimization algorithms in CPLEX

Simplex optimizers
- Primal, dual, network
- LP and QP

Barrier optimizer
- LP, QP, and QCP

Mixed integer optimizers
- Branch & Cut
- Dynamic search
- MIP, MIQP, MIQCP

Remarks:
- The barrier optimizer can explore the presence of multiple threads.
- The barrier optimizer cannot start from an advanced basis, and therefore it has limited application in Branch and Bound methods for MIPLs.
- Re-optimization with the simplex algorithms is faster, when starting from a previous basis.
MIP solvers in CPLEX

Mixed integer optimizers

- Branch & Cut
- Dynamic search
- MIP, MIQP, MIQCP

New algorithm to solve MIPs
- Branch & cut based
- Some user callbacks cannot be used

- IBM trade secret
- Methodology is proprietary
Examples

- **POUTIL** – MILP model from the GAMS library.
- **RHS** – MILP continuous time slot based model for scheduling of continuous processes.
- **RH12** – MILP scheduling model with travelling salesman based constraints.

<table>
<thead>
<tr>
<th></th>
<th>POUTIL</th>
<th>RHS</th>
<th>RH12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equations</td>
<td>2,178</td>
<td>16,886</td>
<td>10,421</td>
</tr>
<tr>
<td>Variables</td>
<td>1,260</td>
<td>12,156</td>
<td>19,134</td>
</tr>
<tr>
<td>0-1 variables</td>
<td>773</td>
<td>5,938</td>
<td>13,340</td>
</tr>
</tbody>
</table>

Computer: machine running Linux, with 8 threads Intel Xeon@ 2.66GHz
Main idea: solve MILP problems by solving a sequence of linear relaxations to provide bounds

MILP formulation

\[ Z(X) = \min \{ cx + fy : (x, y) \in X \} \]

where

\[ X = \{(x, y) \in \mathbb{R}_+^n \times \{0, 1\}^p : Ax + By \geq b\} \]

The relaxation is given by

\[ Z(P_X) = \min \{ cx + fy : (x, y) \in X \} \]

where

\[ P_X = \{(x, y) \in \mathbb{R}_+^n \times [0, 1]^p : Ax + By \geq b\} \]

The linear relaxation provides a lower bound on the optimal objective value:

\[ Z(P_X) \leq Z(X) \]
Remarks
• B&B is not suitable for large scale problems
• The number of iterations grows exponentially with the number of variables

**CPLEX uses the branch and cut algorithm**
• Based on BB
• It is applied to a reformulation of the set V using a pre-processing step and by the addition of cutting planes.
Branch and cut algorithm in CPLEX

Pre-processing

Solve a linear program relaxation

Do all integer variables have integer values?

yes

Update the incumbent

no

Add cuts

Apply heuristics

Choose an integer variable and create two new sub-problems

Select a sub-problem to solve

http://www-01.ibm.com/support/docview.wss?uid=swg21400064
Branch and cut algorithm in CPLEX

Pre-processing

Root node

Solve a linear program relaxation

Do all integer variables have integer values?

yes

Update the incumbent

no

Add cuts

Apply heuristics

http://www-01.ibm.com/support/docview.wss?uid=swg21400064
Pre-processing and probing

• **Goals**
  – Reduce the size of the problem
  – Improve the formulation
    • A new model is defined
    • Tighter formulation without increasing the size of the problem
    • Independent of the relaxation solution

• **Techniques used:**
  – Pre-processing
  – Probing
Pre-processing and probing

• **Pre-processing techniques**
  – Identification of infeasibility
  – Identification of redundancy
  – Improve bounds
  – Rounding (for MIP)

• **Probing techniques**: fix binary variables to either 0 or 1, and check the logical implications
  – Fixing variables
  – Improve coefficients
  – Logical implications

• Both formalized by Savelsbergh (1994) and Wolsey (1998)
Pre-processing example

Initial LP formulation

\begin{align*}
e1.. & \quad z = & 2x1 + x2 - x3; \\
e2.. & \quad 5x1 - 2x2 + 8x3 = & 15; \\
e3.. & \quad 8x1 + 3x2 - x3 = & 9; \\
e4.. & \quad x1 + x2 + x3 = & 6; \\
x1. & \quad \text{up} = & 3; \\
x2. & \quad \text{up} = & 1; \\
x3. & \quad \text{lo} = & 1;
\end{align*}

--- Generating LP model P1
--- wolsey_2.gms(25) 3 Mb
--- 4 rows 4 columns 13 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.017

Cplex 12.2.0.0, GAMS Link 34

Reading data...
Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 4 rows and 4 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec.
LP status(1): optimal

Optimal solution found.
Objective : 3.600000

Final LP formulation

\begin{align*}
e1.. & \quad z = & 2x1 + x2 - x3; \\
e2.. & \quad 5x1 - 2x2 + 8x3 = & 15; \\
e3.. & \quad 8x1 + 3x2 - x3 = & 9; \\
e4.. & \quad x1 + x2 + x3 = & 6; \\
x1. & \quad \text{up} = & 3; \\
x2. & \quad \text{up} = & 1; \\
x3. & \quad \text{lo} = & 1;
\end{align*}
Heuristics at the root node (and afterwards)

Why heuristics?

• Can achieve solutions of difficult MILP problems by exploring parts of the tree that the solver will not.
• May provide good solutions quickly.
• May help to prove optimality
  – explicitly: prune nodes more efficiently
  – Implicitly: provide integer solutions

Types of heuristics:
• Node heuristics: diving
• Neighborhood exploration

Note: heuristic solutions are identified by a ‘+’ in the CPLEX output
Heuristics at the root node (cont).

• Diving heuristics
  1 – Fix a set of integer infeasible variables
  2 – Bound strengthening
  3 – Solve LP relaxation
  4 - Repeat

• Neighborhood
  – Local Branching (LB)
  – *Relaxation Induced Neighborhood Search (RINS)*
  – Guided Dives (GD)
  – *Evolutionary algorithms for polishing MIP solutions*
Cuts and heuristics at the root node

- Example: MILP problem from Wolsey (1998), solved with B&C requiring 3 nodes

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Left</td>
</tr>
<tr>
<td>*</td>
<td>0+    0</td>
</tr>
<tr>
<td>0     0</td>
<td>557.1433</td>
</tr>
<tr>
<td>*</td>
<td>0+    0</td>
</tr>
<tr>
<td>0     0</td>
<td>546.4737</td>
</tr>
<tr>
<td>*</td>
<td>0+    0</td>
</tr>
<tr>
<td>0     0</td>
<td>cutoff</td>
</tr>
</tbody>
</table>

Elapsed real time = 0.08 sec. (tree size = 0.00 MB, solutions = 5)

Clique cuts applied: 1
Cover cuts applied: 7
Zero-half cuts applied: 8
Gomory fractional cuts applied: 1
MIP status(101): integer optimal solution
NEIGHBORHOOD HEURISTICS
• **Idea:** explore the neighborhood of the incumbent to find better solutions

• **Algorithm:**
  – Fix the binary variables with the same values in the continuous relaxation and in the incumbent.
  – Solve a sub-MIP on the remaining variables.

• **Example:**
  – Relaxation: \( x = (0.1, 0, 0, 1, 0.9) \)
  – Incumbent: \( x = (1, 0, 1, 1, 0) \)
  – Fix \( x_2 = 0, x_4 = 1 \)
  – Solve a sub-MIP
• Remarks:
  – It may greatly improve solutions of poor quality
  – Uses the relaxation to define neighborhoods
  – Poor relaxations may lead to large sub-MIP
  – The sub-MIP are not solved optimality
  – It is only invoked every $f$ nodes
• **Idea:** explore the neighborhood of the incumbent by fixing some of the binary variables, and solving a sub-MIP.

• Polishing is based on the integration of an evolutionary algorithm *within* an MIP *branch and bound* framework.

• Can only be called when an incumbent is available.
Integration of EA and B&B

Call solution polishing
Apply evolutionary heuristic
  • Mutation
  • Combination

Fix a subset of the binary variables

Solve a sub-MIP

Is the solution optimal?

YES

Add to the pool.

Update incumbent?

Solution pool
EA operators

EA steps

1. Mutation
   a) Choose a seed from the pool (random)
   b) Fix $f$ variables (apply a random mask)
   c) Solve sub-MIP
   d) Add the solution found to the pool

2. Combination
   a. Choose a pair of solutions from the pool (random)
   b. Fix variables with the same value
   c. Solve the sub-MIP
   d. Add the best solution to the pool

Seed $x=(1, 0, 0, 1, 0)$
New $x=(?, 0, ?, 1, 0)$
Solve a sub-MIP with 2 binary variables.

Seed 1 $x=(1, 0, 0, 1, 0)$
Seed 2 $x=(0, 1, 0, 1, 0)$
New $x=(?, ?, 0, 1, 0)$
Solve a sub-MIP with 2 binary variables.
Relative gap between solution found and best known solution. Bold means better solution.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Initial 50K nodes</th>
<th>After 50% additional time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GD + LB + RINS</td>
<td>Defaults</td>
</tr>
<tr>
<td>glass4</td>
<td>0.34722</td>
<td><strong>0.34722</strong></td>
</tr>
<tr>
<td>liu</td>
<td>0.09747</td>
<td>0.09747</td>
</tr>
<tr>
<td>mkc</td>
<td>0.00020</td>
<td>0.00020</td>
</tr>
<tr>
<td>protfold</td>
<td>0.12903</td>
<td>0.12903</td>
</tr>
<tr>
<td>sp97ar</td>
<td>0.00090</td>
<td>0.00090</td>
</tr>
<tr>
<td>swath</td>
<td>0.02517</td>
<td>0.02517</td>
</tr>
<tr>
<td>timtab2</td>
<td>0.07545</td>
<td>0.07545</td>
</tr>
<tr>
<td>bg512142</td>
<td>0.04287</td>
<td>0.04287</td>
</tr>
<tr>
<td>dg012142</td>
<td>0.26215</td>
<td>0.26215</td>
</tr>
<tr>
<td>B2C1S1</td>
<td>0.00707</td>
<td>0.00707</td>
</tr>
<tr>
<td>pharma1</td>
<td>0.00288</td>
<td>0.00288</td>
</tr>
<tr>
<td>sp97ic</td>
<td>0.00360</td>
<td>0.00360</td>
</tr>
<tr>
<td>sp98ar</td>
<td>0.00083</td>
<td>0.00083</td>
</tr>
<tr>
<td>sp98ic</td>
<td>0.00289</td>
<td>0.00289</td>
</tr>
<tr>
<td>UMIS</td>
<td>0.00107</td>
<td>0.00107</td>
</tr>
<tr>
<td>rococoB10-011001</td>
<td>0.02917</td>
<td>0.02328</td>
</tr>
<tr>
<td>rococoB11-110001</td>
<td>0.03058</td>
<td>0.03058</td>
</tr>
<tr>
<td>rococoB12-111111</td>
<td>0.02919</td>
<td>0.02919</td>
</tr>
<tr>
<td>rococoC10-100001</td>
<td>0.06025</td>
<td>0.06025</td>
</tr>
<tr>
<td>rococoC11-010100</td>
<td>0.04050</td>
<td>0.01249</td>
</tr>
<tr>
<td>rococoC12-100000</td>
<td>0.01349</td>
<td><strong>0.01349</strong></td>
</tr>
<tr>
<td>rococoC12-111100</td>
<td>0.01033</td>
<td>0.01033</td>
</tr>
<tr>
<td>lj2b</td>
<td>0.01574</td>
<td>0.01574</td>
</tr>
<tr>
<td>lj7</td>
<td>0.24904</td>
<td>0.24904</td>
</tr>
<tr>
<td>lj9</td>
<td>0.77430</td>
<td>0.53763</td>
</tr>
<tr>
<td>lj10</td>
<td>0.03254</td>
<td>0.03254</td>
</tr>
<tr>
<td>lj12</td>
<td>0.32932</td>
<td>0.23932</td>
</tr>
</tbody>
</table>
Solution polishing remarks

• Requires at least one solution
• Keeps the logic of the lower and upper bound used in B&B.

• Solution polishing can be activated after:
  – Node limit
  – Time limit
  – Within a gap %
Impact of cutting planes and heuristics

- Upper bound
  - LP
- Lower bound
- Cutting planes
- Sub-MIP
- Heuristics
Parallel optimizers in CPLEX

• Parallelization available:
  – MIP solver
  – Barrier algorithm
  – Concurrent optimization

• Concurrent optimization for solving LP and QP
  – CPLEX launches several optimizers to solve the same problem, the process terminates when the first solver stops:
    • Thread 1 - dual simplex
    • Thread 2 - barrier.
    • Thread 3 – primal simplex
    • Thread >3 - barrier run.
Parallelization in the B&B
  - Solution of the root node
  - Solution of nodes
  - Strong branching in parallel

2 modes are available:
  - Deterministic – invariance and repeatability of the search path and results
  - Opportunistic – each run may lead to a different search path and results – usually out-performs the deterministic

Which one should be used?
Log of the parallelization

- **Deterministic**
  
  Root node processing (before b&c):
  
  Real time = 37.31

  Parallel b&c, 8 threads:
  
  Real time = 3565.95
  Sync time (average) = 93.98
  **Wait time (average) = 216.70**

  -------

  Total (root+branch&cut) = 3603.26 sec.

- **Opportunistic**

  Root node processing (before b&c):
  
  Real time = 34.47

  Parallel b&c, 8 threads:
  
  Real time = 3566.18
  Sync time (average) = 5.97
  **Wait time (average) = 4.76**

  -------

  Total (root+branch&cut) = 3600.65 sec.
Example: POUTIL

<table>
<thead>
<tr>
<th>RMIP root</th>
<th>246,984.7</th>
</tr>
</thead>
</table>

**CPLEX 12.2**

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>RMIP</td>
</tr>
<tr>
<td>1</td>
<td>950</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>4D</td>
<td>211</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MIP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>266,793.0</td>
</tr>
</tbody>
</table>
# Example RH12

<table>
<thead>
<tr>
<th>RMIP root</th>
<th>5,225,207</th>
</tr>
</thead>
</table>

## CPLEX 12.0

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>Objective function</th>
<th>MIP</th>
<th>MIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,600</td>
<td>101.2</td>
<td>5,166,820</td>
<td></td>
<td><strong>-444,529,600</strong></td>
</tr>
<tr>
<td>4D</td>
<td>3,600</td>
<td>114.8</td>
<td>5,165,611</td>
<td></td>
<td><strong>-34,831,279</strong></td>
</tr>
<tr>
<td>4O</td>
<td>3,600</td>
<td>10.5</td>
<td>5,166,242</td>
<td></td>
<td><strong>4,674,076</strong></td>
</tr>
<tr>
<td>8D</td>
<td>3,600</td>
<td>42</td>
<td>5,166,870</td>
<td></td>
<td>3,639,156</td>
</tr>
<tr>
<td>8O - 1st run</td>
<td>3,600</td>
<td>1124.5</td>
<td>5,165,035</td>
<td></td>
<td><strong>-504,162</strong></td>
</tr>
<tr>
<td>8O - 2nd run</td>
<td>3,600</td>
<td>17.1</td>
<td>5,168,434</td>
<td></td>
<td><strong>4,412,006</strong></td>
</tr>
</tbody>
</table>
Effect of parallelization and polishing

![Graph showing the effect of parallelization and polishing with objective function vs. number of nodes. The graph compares different optimization techniques such as RHS 1D, RHS 4D, RHS 8D, RHS 4O, RHS 8O, RHS POL 8D, and Without heuristics 8D. The x-axis represents the number of nodes, ranging from 0 to 120,000, while the y-axis shows the objective function values ranging from -1E+11 to 2E+10.]
Effect of parallelization and polishing

Objective function

# Nodes

-1.0E+08 to -1.0E+08

0 to 80,000

RHS 1D
RHS 4D
RHS 8D
RHS 4O
RHS 8O
RHS POL 8D
Effect of parallelization and polishing

![Graph showing the effect of parallelization and polishing on the objective function.](image)

- Objective function
- # Nodes
- RHS 1D
- RHS 4D
- RHS 8D
- RHS 4O
- RHS 8O
- RHS POL 8D

12/07/2010  EWO seminar
Impact of the solution polish option

Objective function vs. CPU time (s)

- RHS 8D POL
- RHS 8D
Ineffective solution polishing

<table>
<thead>
<tr>
<th></th>
<th>RMIP root</th>
<th>246,984.7</th>
</tr>
</thead>
</table>

**CPLEX 12.2**

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>Objective function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMIP</td>
<td></td>
<td></td>
<td>266,793.0</td>
</tr>
<tr>
<td>MIP</td>
<td></td>
<td></td>
<td>266,793.0</td>
</tr>
<tr>
<td>1</td>
<td>950</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>4D</td>
<td>211</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>4O</td>
<td>206</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>8D</td>
<td>95</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>8O</td>
<td>61</td>
<td>0.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>8D Polishing</td>
<td>1000</td>
<td>0.94</td>
<td>264,291.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>266,793.0</td>
</tr>
</tbody>
</table>
MIP start

- CPLEX has the option to start from a user-defined solution
  - The solution can be feasible or unfeasible
  - If the solution is not feasible, CPLEX uses a heuristic to try to repair the solution
    - Helps to find a feasible solution
  - If the solution is feasible, heuristics such as RINS or solution polishing can be used
  - Useful to debug a model
Integration of MIP start and polishing

- Solve MILP
  - Max time: 300s
  - Solution Polishing after 60s
  - Turn off cuts

- Feed solution to solver
  - Turn on cuts

- Solve MILP
  - Solution polishing to improve solution
  - > 90% time

→ Avoid time generating cuts

 Restart, and use a starting solution to improve the search.
### RHS results: polishing and MIP start

#### CPLEX 12.2

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>RMIP</th>
<th>MIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3600</td>
<td>3.4</td>
<td>2,669.0</td>
<td>2,580.5</td>
</tr>
<tr>
<td>4D</td>
<td>3600</td>
<td>3.3</td>
<td>2,667.5</td>
<td>2,582.4</td>
</tr>
<tr>
<td>4O</td>
<td>3600</td>
<td>2.3</td>
<td>2,667.2</td>
<td>2,607.2</td>
</tr>
<tr>
<td>8D</td>
<td>3600</td>
<td>3.4</td>
<td>2,666.3</td>
<td>2,578.8</td>
</tr>
<tr>
<td>8O</td>
<td>3600</td>
<td>2.3</td>
<td>2,665.9</td>
<td>2,607.2</td>
</tr>
<tr>
<td>8D P - 60s</td>
<td>3600</td>
<td>2.2</td>
<td>2,668.8</td>
<td>2,610.4</td>
</tr>
<tr>
<td>8D Start</td>
<td>3600</td>
<td>2.0</td>
<td>2,656.6</td>
<td>2,603.5</td>
</tr>
<tr>
<td>CPLEX 7.1</td>
<td>3600</td>
<td>-</td>
<td>2,687.9</td>
<td>-</td>
</tr>
</tbody>
</table>
Solution pools

• Motivation:
  – Value on having more than one solution
  – Model does not capture the full essence of the process
  – Approximations on creating the model
  – Data is not accurate

• Goal: generate and keep multiple solution
  – MIP, MIQCP

• Options and tools:
  – Collect solutions with a given percentage of the optimal solution
  – Collect diverse solutions
  – Collect solutions with diverse properties
  – Difficult to implement with rolling horizon decompositions
Solution pools (cont.)

- Example of application (Emilie Danna, CPLEX)

Remark: difficult to implement with rolling horizon decompositions
Tuning tool

• Motivation
  – MIP solvers have multiple algorithm parameters
  – The performance of the solver depends on these parameters
  – Default values in solvers are defined in order to work well for a large collection of problems
    • May not work for the user specific problem

• Goal: identify the solver parameters that improve the performance of the solver for a given set of problems.
## Tuning tool: example

### CPLEX 12.2

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>Objective function</th>
<th>RMIP</th>
<th>MIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>949</td>
<td>0.0</td>
<td></td>
<td>266,793.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>8D</td>
<td>95</td>
<td>0.0</td>
<td></td>
<td>266,793.0</td>
<td>266,793.0</td>
</tr>
</tbody>
</table>

Apply the tuning tool
Time = 327s

- threads 8
- cutpass=-1
- heurfreq=-1
- itlim=100000000
- parallelmode=1
- probe=-1
- varsel=4

### CPLEX 12.2

<table>
<thead>
<tr>
<th>Threads</th>
<th>CPU time (s)</th>
<th>Gap (%)</th>
<th>Objective function</th>
<th>RMIP</th>
<th>MIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>67</td>
<td>0.0</td>
<td></td>
<td>266,793.0</td>
<td>266,793.0</td>
</tr>
<tr>
<td>8D</td>
<td>8</td>
<td>0.0</td>
<td></td>
<td>266,793.0</td>
<td>266,793.0</td>
</tr>
</tbody>
</table>
Variability in the performance may occur in CPLEX 12.2 due to:
- Opportunistic parallelization
- Heuristics: polishing option (random seed)
- Numerical reasons

Variability may occur on:
- Computational time
- Performance in terms of nodes, iterations
- Quality of the solution

Remarks:
- It seems particularly relevant when optimality cannot be guaranteed within the maximum time set.
- If repeatability of the results is required the above options should not be used, mainly in the development phase.
- However, it is an opportunity to obtain better solutions.
Final remarks

• The increasing performance of CPLEX has been allowing us to solve more complex problems.

• The CPLEX default parameters may not be a good choice for all problems.

• The solution pool may be an important feature to implement some decompositions.

• Topics not discussed:
  – Infeasibility analysis tool
  – Interface of CPLEX with other applications and programming languages
  – Comparison of the CPLEX performance with other solvers
  – Use of callbacks
• Technical support from IBM ILOG: “CPLEX Performance Tuning for Mixed Integer Programs”

• Approach to tune CPLEX for MILPs
  1. Use a good formulation.
  2. Solve with default values.
  3. Check the CPLEX log to evaluate:
     a) if it is difficult to find the first integer solution.
     b) the progress of the lower and upper bound, and determine if it is difficult to obtain integer solutions.
  4. Diversify or change the search path:
     a) Set priorities for the variables.
     b) Increase the frequency of the use of heuristics if it is difficult to find integer solutions.
     c) Use the polishing option to improve the incumbent. When the polishing option is activated, CPLEX will spend more time solving sub-MIPs, and little progress is made on the relaxation.
     d) Use the parallel mode with the opportunistic option.
     e) Change the branching strategy
  5. Improve the linear relaxation solution
     a) Increase the level of generation of cuts (increases the computational times)
     b) Increase the level of probing (increases the computational times)
  6. If the goal is to decrease the computational time, turn off heuristics and turn off the generation of cutting planes, it may be faster.
  7. Use the tuning tool.
References for CPLEX and MIP

- **CPLEX manuals**
  - IBM ILOG CPLEX Manual

- **Presolve and conflict analysis**
  - Rothberg, E., ILOG, Inc. The CPLEX Library: Presolve and Cutting Planes
• **Branch and bound and LP**
  – Rothberg E., ILOG, Inc. The CPLEX Library: Mixed Integer Programming
  – Rothberg, E., ILOG, Inc. The CPLEX Library: Presolve and Cutting Planes

• **Local search heuristics**
  – Rothberg, E. ILOG, Inc. The CPLEX Library: MIP Heuristics
References (cont.)

- **Local search heuristics (cont.)**

- **Parallelization**
Other software packages

• Commercial
  – XPRESS, FICO
  – XA, Sunset Software Technology
  – MOSEK, MOSEK
  – GUROBI, GUROBI Optimization

• Non-commercial
  – SCIP, ZIB
  – MINTO, CORAL
  – GLPK, GNU
  – CBC, COIN-OR
  – SYMPHONY, COIN_OR

• Benchmark sites:
  – http://miplib.zib.de
Example

• Consider the pure integer programming problem:

$$\begin{align*}
\min z &= -5y_1 - 2y_2 \\
st. \\
- y_1 + y_2 &\leq 10 \\
2y_1 + y_2 &\leq 15 \\
8y_1 + y_2 &\leq 30 \\
y_1, y_2 &\in \mathbb{Z}_+
\end{align*}$$
Initialization
$L = \{P_x\}$
$\overline{Z} := +\infty$

Branching
when $Z(V) \leq \overline{Z}$ and $y^V_j \notin \mathbb{Z}$
select branching variable $y^V_j \notin \mathbb{Z}$
set $L := L \cup \{V^0, V^1\}$ where
$V^0 = V \cap \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^p_+ : y_j \leq \lfloor y^V_j \rfloor\}$
$V^1 = V \cap \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{R}^p_+ : y_j \leq \lceil y^V_j \rceil\}$
GO TO Termination

\begin{align*}
\min z &= -5y_1 - 2y_2 \\
\text{st.} &y_1 + y_2 \leq 10 \\
&2y_1 + y_2 \leq 15 \\
&8y_1 + y_2 \leq 30 \\
&y_1 \leq 2 \\
&y_1, y_2 \in \mathbb{Z}_+
\end{align*}

\begin{align*}
\min z &= -5y_1 - 2y_2 \\
\text{st.} &y_1 + y_2 \leq 10 \\
&2y_1 + y_2 \leq 15 \\
&8y_1 + y_2 \leq 30 \\
&y_1 \geq 3 \\
&y_1, y_2 \in \mathbb{Z}_+
\end{align*}
Initialization
\[ L = \{ P_x \} \]
\[ \overline{Z} := +\infty \]

Node selection and solve
Select \( V \in L \) and let \( L := L \setminus \{ V \} \)
Compute \( Z(V), (x^V, y^V) \)

\[ Z = -27.0 \]
\[ Z = -32.0 \]
\[ Z = -32.5 \]

Upper bound
Lower bound
Cuts and heuristics at the root node

- **Given:** is a vector of variables $x \in \{0,1\}^p$ that by optimality can be treated as continuous, to $x \in [0,1]^p$.

- **Question:** what is the impact of relaxing the variables? (number of variables, relaxation, search)

**Example**

In the RHS model the binary variables $Z_{i,l,m,t}$ and $TRT_{i,k,m,t}$ can be relaxed to continuous variables

Reduction of the number of binary variables: 5581 to 1502.
LP solution and relaxation

- LP solution is the same for both models
  Optimal solution found.
  Objective: 2692.510176

- However, the LP relaxation is different at the beginning of the root node iterations.

**CPLEX log using Z and TRT as continuous variables**

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Value</th>
<th>Objective</th>
<th>Cuts:</th>
<th>Elapsed real time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2690.3084</td>
<td>2690.3084</td>
<td>9175</td>
<td>24.64 sec.</td>
</tr>
<tr>
<td>0</td>
<td>2688.2465</td>
<td></td>
<td>11483</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2687.0382</td>
<td></td>
<td>13859</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.7985</td>
<td></td>
<td>14924</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.6539</td>
<td></td>
<td>15602</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.5623</td>
<td></td>
<td>15957</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.5612</td>
<td></td>
<td>16028</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.5612</td>
<td></td>
<td>16073</td>
<td></td>
</tr>
<tr>
<td>Heuristic still looking.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2686.5612</td>
<td>2686.5612</td>
<td>16073</td>
<td></td>
</tr>
</tbody>
</table>

Elapsed real time = 24.64 sec. (tree size = 0.01 MB, solutions = 0)
**LP solution and relaxation**

### CPLEX log using $Z$ and $TRT$ as binary variables

<table>
<thead>
<tr>
<th>BIN</th>
<th>2692.1693</th>
<th>1661</th>
<th>2692.1693</th>
<th>11471</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONT</td>
<td>2689.1996</td>
<td>1511</td>
<td>Cuts: 365</td>
<td>14327</td>
</tr>
<tr>
<td>CONT</td>
<td>2684.7527</td>
<td>1567</td>
<td>Cuts: 378</td>
<td>16553</td>
</tr>
<tr>
<td>CONT</td>
<td>2683.4370</td>
<td>1490</td>
<td>Cuts: 263</td>
<td>19210</td>
</tr>
<tr>
<td>CONT</td>
<td>2682.3135</td>
<td>1484</td>
<td>Cuts: 169</td>
<td>20982</td>
</tr>
<tr>
<td>CONT</td>
<td>2681.2411</td>
<td>1595</td>
<td>Cuts: 143</td>
<td>24554</td>
</tr>
<tr>
<td>CONT</td>
<td>2679.2076</td>
<td>1467</td>
<td>Cuts: 119</td>
<td>26157</td>
</tr>
<tr>
<td>CONT</td>
<td>2678.3433</td>
<td>1378</td>
<td>Cuts: 157</td>
<td>28551</td>
</tr>
<tr>
<td>CONT</td>
<td>2677.9695</td>
<td>1374</td>
<td>Cuts: 109</td>
<td>29187</td>
</tr>
<tr>
<td>CONT</td>
<td>2677.5116</td>
<td>1438</td>
<td>Cuts: 68</td>
<td>31526</td>
</tr>
<tr>
<td>CONT</td>
<td>2677.3114</td>
<td>1455</td>
<td>Cuts: 77</td>
<td>32456</td>
</tr>
<tr>
<td>CONT</td>
<td>2677.1595</td>
<td>1397</td>
<td>Cuts: 40</td>
<td>32775</td>
</tr>
<tr>
<td>CONT</td>
<td>2676.8246</td>
<td>1373</td>
<td>Cuts: 54</td>
<td>33240</td>
</tr>
<tr>
<td>CONT</td>
<td>2676.4693</td>
<td>1442</td>
<td>Cuts: 54</td>
<td>34183</td>
</tr>
</tbody>
</table>

**RMIP root**

<table>
<thead>
<tr>
<th>RMIP</th>
<th>Beginning</th>
<th>End</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIN</td>
<td>2,693</td>
<td>2,692</td>
<td>2,676</td>
</tr>
<tr>
<td>CONT</td>
<td>2,693</td>
<td>2,690</td>
<td>2,690</td>
</tr>
</tbody>
</table>

**Elapased real time**

|Elapsed real time| 61105| 47491| 2665.6594| 1208| 2578.7965| 2666.3089| 30510359| 3.39%|

- The initial LP relaxations at the root node are different
- The solutions at the end of the root node are different: 2686.5612 vs 2676.4693
- The final relaxation is better when using binary variables

12/07/2010 EWO seminar
Heuristics motivational example

- RHS problem optimized with heuristics and heuristics turned off.

Without heuristics the solution is not close to the best known solution.

Without heuristics a feasible solution is found much later.
### Heuristics automatic

<table>
<thead>
<tr>
<th>Node</th>
<th>Objective</th>
<th>Time</th>
<th>Cuts:</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>499</td>
<td>385</td>
<td>2674.4232</td>
<td>1298</td>
<td>2676.4137</td>
<td>511018</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Elapsed real time = **86.54 sec.** (tree size = 3.99 MB, solutions = 0)

* 604+ 321 -1.78665e+10 2672.4010 570690 100.00%
604 322 2671.7797 1341 -1.78665e+10 2671.7797 577744 100.00%
605 323 2671.5395 1410 -1.78665e+10 2671.7797 582540 100.00%
608 324 2665.7742 1321 -1.78665e+10 2671.5025 589020 100.00%
620 331 2670.9349 1440 -1.78665e+10 2671.2627 604374 100.00%

* 658+ 247 -1.46007e+10 2671.2627 662376 100.00%

### Heuristics turned off

<table>
<thead>
<tr>
<th>Node</th>
<th>Objective</th>
<th>Time</th>
<th>Cuts:</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>9656</td>
<td>8295</td>
<td>2668.7217</td>
<td>1328</td>
<td>2669.6422</td>
<td>3285646</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Elapsed real time = **401.82 sec.** (tree size = 824.86 MB, solutions = 0)

Nodefile size = 673.26 MB (610.47 MB after compression)

* 11283 6532 integral 0 -9.18449e+10 2669.6422 3649047 100.00%