IBM Software Group

Performance Tuning Using CPLEX Optimization Studio IDE

Guang Feng (gfeng@us.ibm.com)
Nikhila Arkalgud (narkalgu@us.ibm.com)
Technical Support Engineers, Level 2
10 July 2013

WebSphere® Support Technical Exchange

ON DEMAND BUSINESS™

Agenda

- Introduction to CPLEX Optimization Studio IDE
- Performance Tuning using the Profiler Tool
- Performance Tuning using the Tuning Tool
- Some Other Features of the IDE
- Installing CPLEX Optimization Studio on Windows

Introduction to CPLEX Optimization Studio IDE

Optimization Programming Language (OPL)

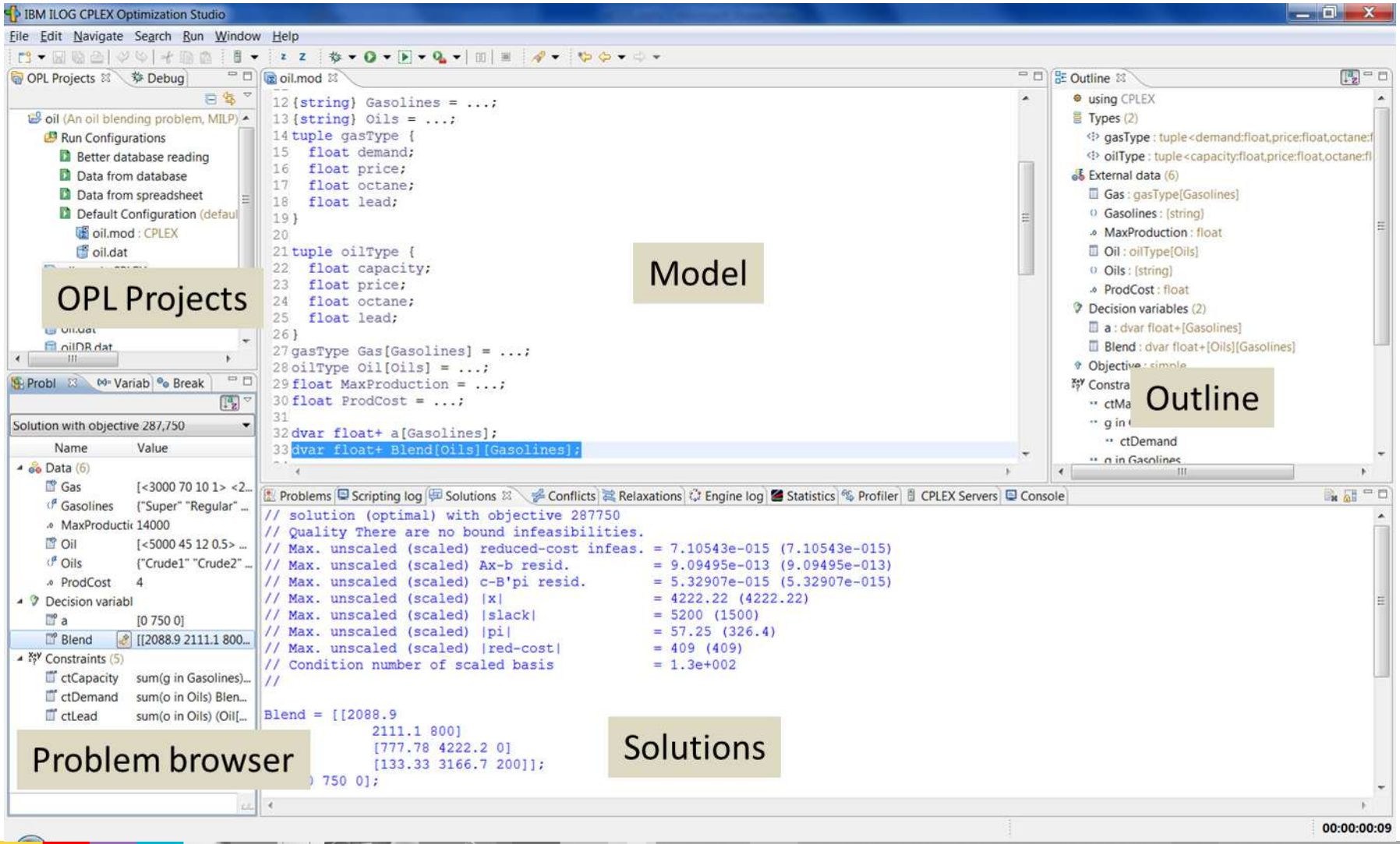
- A programming language specifically designed for building optimization models
- Provides a clear separation between Input Data and Model
- Provides easy access to connect to different databases

```
//Data Initialization
{string} Products = ...;
{string} Components = ...;
float Demand[Products][Components] = ...;
float Profit[Products] = ...;
float Stock[Components] = ...;
//Decision variables
dvar float+ Production[Products];
//Objective Function
maximize
sum( p in Products )Profit[p] *
Production[p];
//Constraints
subject to {
forall( c in Components )
ct:sum( p in Products )Demand[p][c] *
Production[p] <= Stock[c];
}
```

CPLEX Optimization Studio IDE

- CPLEX Optimization Studio IDE is an Eclipse based Development IDE used to create Optimization models written in OPL
- The development studio directly supports solvers CPLEX and CP Optimizer
- Provides access to connect to other applications such as SPSS Modeler
- Provides option to perform remote solves
- Provides Debugging tools

CPLEX Optimization Studio IDE

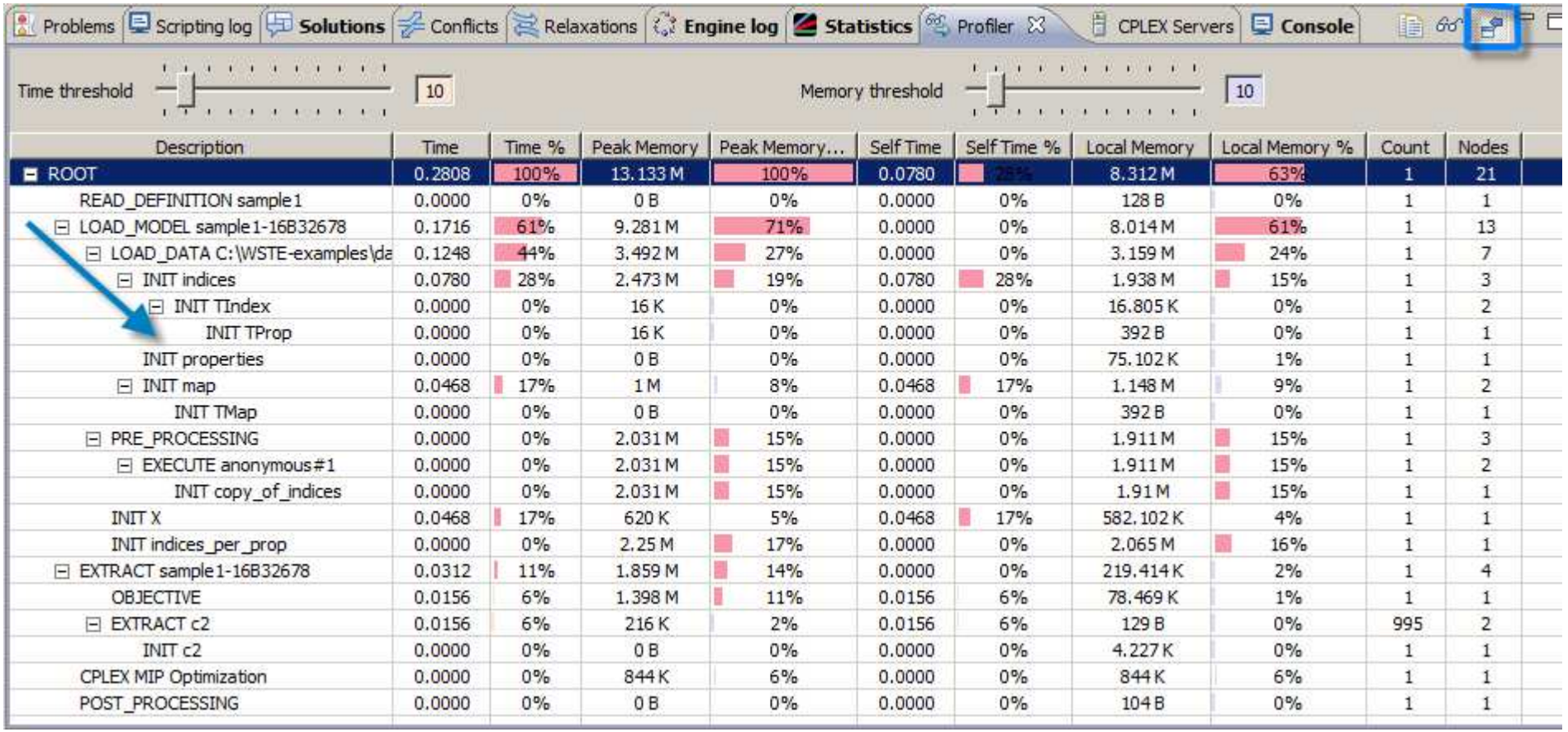


Performance Tuning using the Profiler Tool

What is the Profiler

- The Profiler is an OPL feature that collects statistics of memory and time spent for model generation and model optimization.
- The tool is available in IDE and command line
  - The IDE can highlight the cells exceeding certain threshold (the columns showing percentage)
    - Threshold values are adjustable independently
  - For platforms without IDE, oplrun has an option -profile
    - Also available in API IloOplProfiler
    - Linux users should try to use the IDE first
- Profiler outputs are populated automatically in Profiler tab
  - Enabled by default
  - IDE feature save results can save profiling results for further analysis.
  - You can also copy to clipboard

The Profiler Tab



Description	Time	Time %	Peak Memory	Peak Memory...	Self Time	Self Time %	Local Memory	Local Memory %	Count	Nodes
ROOT	0.2808	100%	13.133 M	100%	0.0780	28%	8.312 M	63%	1	21
READ_DEFINITION sample1	0.0000	0%	0 B	0%	0.0000	0%	128 B	0%	1	1
LOAD_MODEL sample1-16B32678	0.1716	61%	9.281 M	71%	0.0000	0%	8.014 M	61%	1	13
LOAD_DATA C:\WSTE-examples\da	0.1248	44%	3.492 M	27%	0.0000	0%	3.159 M	24%	1	7
INIT indices	0.0780	28%	2.473 M	19%	0.0780	28%	1.938 M	15%	1	3
INIT TIndex	0.0000	0%	16 K	0%	0.0000	0%	16.805 K	0%	1	2
INIT TProp	0.0000	0%	16 K	0%	0.0000	0%	392 B	0%	1	1
INIT properties	0.0000	0%	0 B	0%	0.0000	0%	75.102 K	1%	1	1
INIT map	0.0468	17%	1 M	8%	0.0468	17%	1.148 M	9%	1	2
INIT TMap	0.0000	0%	0 B	0%	0.0000	0%	392 B	0%	1	1
PRE_PROCESSING	0.0000	0%	2.031 M	15%	0.0000	0%	1.911 M	15%	1	3
EXECUTE anonymous#1	0.0000	0%	2.031 M	15%	0.0000	0%	1.911 M	15%	1	2
INIT copy_of_indices	0.0000	0%	2.031 M	15%	0.0000	0%	1.91 M	15%	1	1
INIT X	0.0468	17%	620 K	5%	0.0468	17%	582.102 K	4%	1	1
INIT indices_per_prop	0.0000	0%	2.25 M	17%	0.0000	0%	2.065 M	16%	1	1
EXTRACT sample1-16B32678	0.0312	11%	1.859 M	14%	0.0000	0%	219.414 K	2%	1	4
OBJECTIVE	0.0156	6%	1.398 M	11%	0.0156	6%	78.469 K	1%	1	1
EXTRACT c2	0.0156	6%	216 K	2%	0.0156	6%	129 B	0%	995	2
INIT c2	0.0000	0%	0 B	0%	0.0000	0%	4.227 K	0%	1	1
CPLEX MIP Optimization	0.0000	0%	844 K	6%	0.0000	0%	844 K	6%	1	1
POST_PROCESSING	0.0000	0%	0 B	0%	0.0000	0%	104 B	0%	1	1

How to analyze performance issues from scratch using the Profiler Tool?

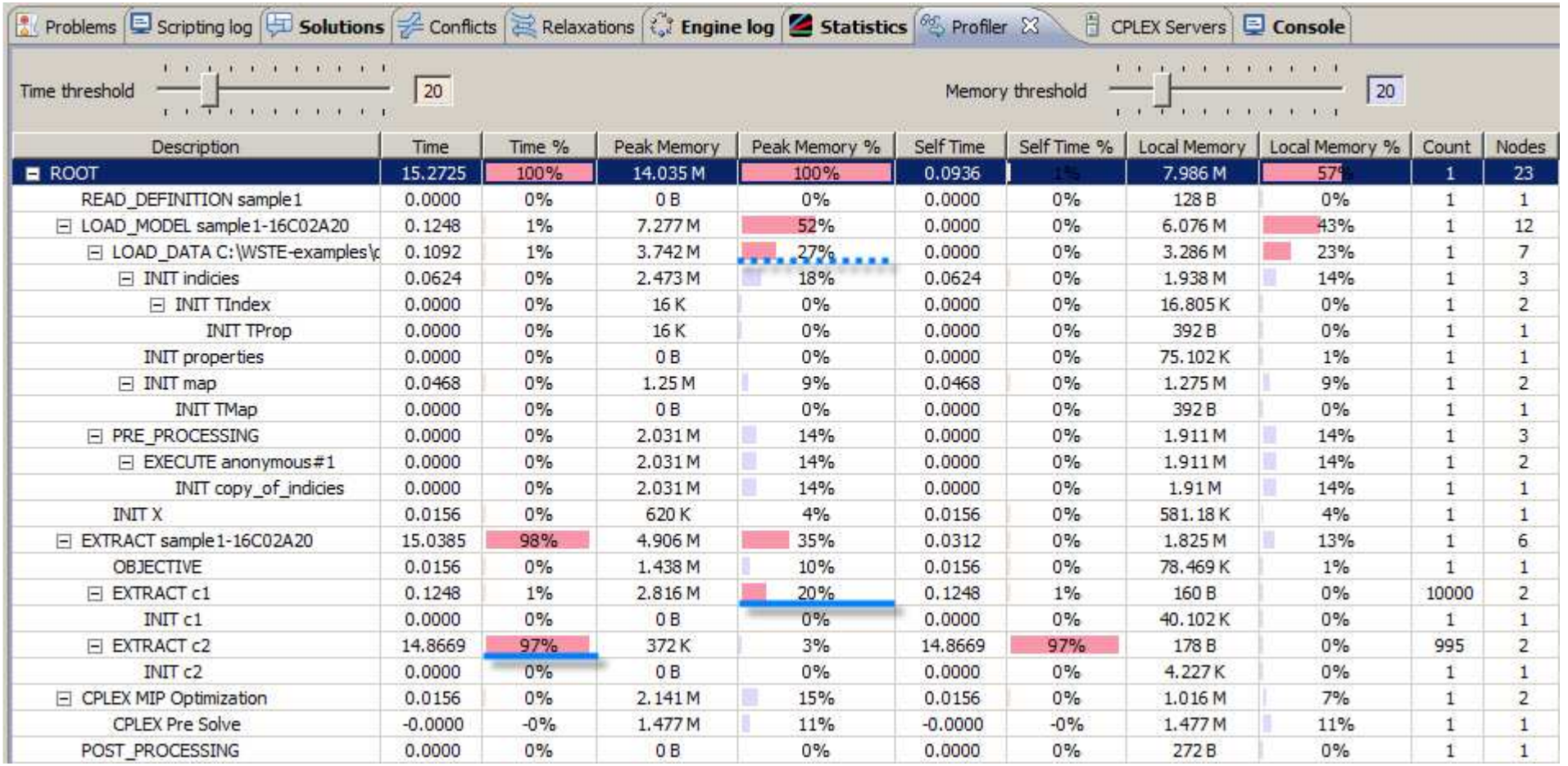
- Set a threshold to focus on potentially less efficient codes up to a certain degree
- Perform a run with profiler enabled to understand the current situation
- Look into the results and see if the performance is reasonable or not.
  - Focus on issues with most significant impacts
  - What is the computational complexity of your formulation and data?
  - Are the profiler outputs matching your estimates?
  - Can you improve the formulation/coding with lower complexity?
- Repeat the process with smaller threshold value, if needed.

Our Sample

```
include "common.mod";

{TIndex} indices = ...;

{TIndex} copy_of_indices = {ind | ind in indices};

execute {
  copy_of_indices; // simulating temporary data usage
}

{TProp} properties = ...;

{TMap} map with prop in properties, ind in indices= ...;

dvar int+ X[indices];

minimize sum(ind in indices) X[ind];

subject to {

  forall(ind in indices)
       c1: if (ind.weight > 150000) X[ind] >= 1;

  forall(prop in properties) {
    c2: sum(ind in indices: <ind, prop> in map) X[ind] <=
3000;
  }
}
```

```
tuple TProp {
  string prop;
  int limit;
}

tuple TIndex {
  int i;
  TProp prop1;
  TProp prop2;
  TProp prop3;
  int weight;
};

tuple TMap {
  TIndex ind;
  TProp prop;
}
```

Initial Profiling run with default threshold (20%)



Problems | Scripting log | Solutions | Conflicts | Relaxations | Engine log | Statistics | Profiler | CPLEX Servers | Console

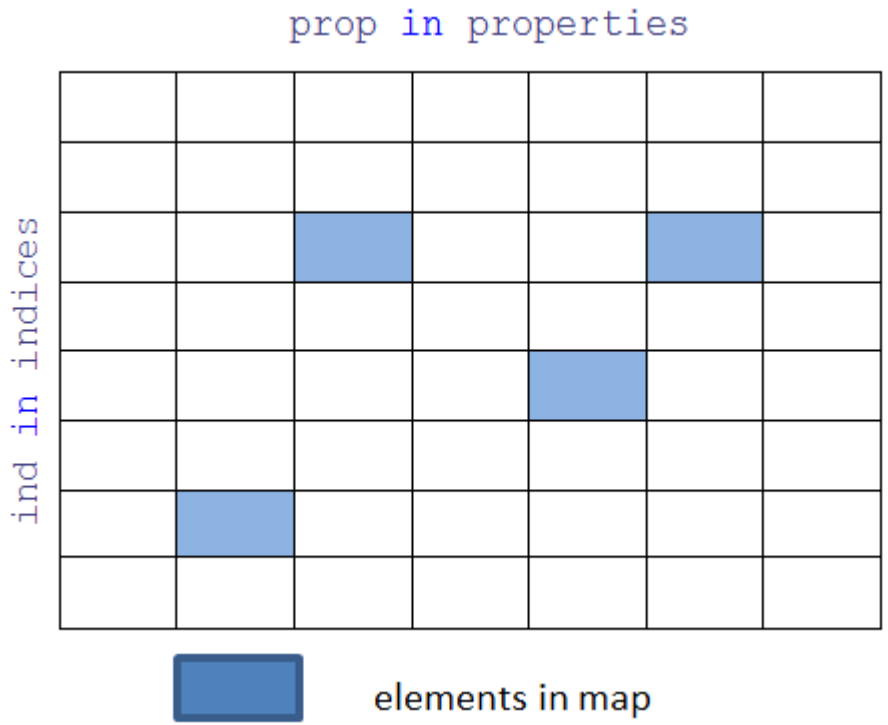
Time threshold 20 Memory threshold 20

Description	Time	Time %	Peak Memory	Peak Memory %	Self Time	Self Time %	Local Memory	Local Memory %	Count	Nodes
ROOT	15.2725	100%	14.035 M	100%	0.0936	1%	7.986 M	57%	1	23
READ_DEFINITION sample1	0.0000	0%	0 B	0%	0.0000	0%	128 B	0%	1	1
LOAD_MODEL sample1-16C02A20	0.1248	1%	7.277 M	52%	0.0000	0%	6.076 M	43%	1	12
LOAD_DATA C:\WSTE-examples\c	0.1092	1%	3.742 M	27%	0.0000	0%	3.286 M	23%	1	7
INIT indicies	0.0624	0%	2.473 M	18%	0.0624	0%	1.938 M	14%	1	3
INIT TIndex	0.0000	0%	16 K	0%	0.0000	0%	16.805 K	0%	1	2
INIT TProp	0.0000	0%	16 K	0%	0.0000	0%	392 B	0%	1	1
INIT properties	0.0000	0%	0 B	0%	0.0000	0%	75.102 K	1%	1	1
INIT map	0.0468	0%	1.25 M	9%	0.0468	0%	1.275 M	9%	1	2
INIT TMap	0.0000	0%	0 B	0%	0.0000	0%	392 B	0%	1	1
PRE_PROCESSING	0.0000	0%	2.031 M	14%	0.0000	0%	1.911 M	14%	1	3
EXECUTE anonymous#1	0.0000	0%	2.031 M	14%	0.0000	0%	1.911 M	14%	1	2
INIT copy_of_indicies	0.0000	0%	2.031 M	14%	0.0000	0%	1.91 M	14%	1	1
INIT X	0.0156	0%	620 K	4%	0.0156	0%	581.18 K	4%	1	1
EXTRACT sample1-16C02A20	15.0385	98%	4.906 M	35%	0.0312	0%	1.825 M	13%	1	6
OBJECTIVE	0.0156	0%	1.438 M	10%	0.0156	0%	78.469 K	1%	1	1
EXTRACT c1	0.1248	1%	2.816 M	20%	0.1248	1%	160 B	0%	10000	2
INIT c1	0.0000	0%	0 B	0%	0.0000	0%	40.102 K	0%	1	1
EXTRACT c2	14.8669	97%	372 K	3%	14.8669	97%	178 B	0%	995	2
INIT c2	0.0000	0%	0 B	0%	0.0000	0%	4.227 K	0%	1	1
CPLEX MIP Optimization	0.0156	0%	2.141 M	15%	0.0156	0%	1.016 M	7%	1	2
CPLEX Pre Solve	-0.0000	-0%	1.477 M	11%	-0.0000	-0%	1.477 M	11%	1	1
POST_PROCESSING	0.0000	0%	0 B	0%	0.0000	0%	272 B	0%	1	1

Performance Analysis of “EXTRACT c2” (97% of runtime)

- Related Codes

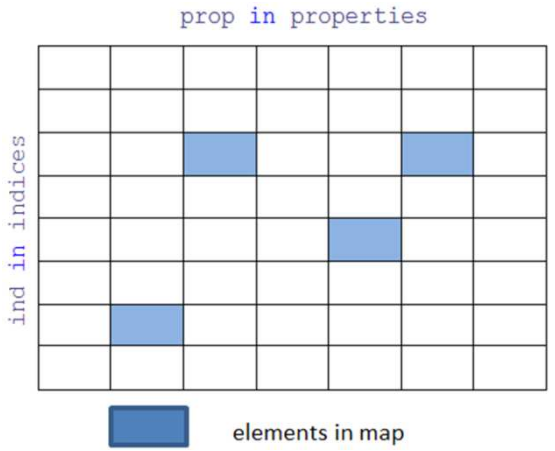
```
tuple TMap {
  TIndex ind;
  TProp prop;
}

{TMap} map with prop in
  properties, ind in indices=
  ...;

forall(prop in properties) {
    c2: sum(ind in indices:
  <ind, prop> in map) X[ind] <=
  3000;
   }
```



Performance Analysis of “EXTRACT c2” (97% of runtime)

- Related Codes

```
forall(prop in properties) {
    c2: sum(ind in indices:
        <ind, prop> in map) X[ind] <= 3000;
  }
```



- Analysis
  - All possible values of tuple element `ind` are from set `indices` (the `with` keyword)
  - So it is not necessary to traverse the set `indices` if its cardinality is larger than `map`

Performance Analysis of “EXTRACT c2” (97% of runtime)

- Improved formulation

```
forall(prop in properties) {
   c2: sum(m_item in map: m_item.prop ==  prop) X[m_item.ind] <= 3000;
 }
```

or

```
{TIndex} indices_per_prop [properties];

execute {
  for(var m_item in map) {
 //addOnly is faster than add, if you don't need a handle
 //to the object added
 indices_per_prop[m_item.prop].addOnly(m_item.ind);
  }
}
…
forall(prop in properties) {
   c2: sum(ind in indices_per_prop[prop]) X[ind] <= 3000;
 }
```

Performance Analysis of "EXTRACT c1" (20% of memory)

- Related Codes

```
forall(ind in indices)
   c1: if (ind.weight > 150000) X[ind] >= 1;
```

- Analysis
  - This formulation creates a labeled constraint no matter the condition of the `if` statement is true or not.
    - Inefficient if the condition is false for most `ind` values
  - So it is not necessary to traverse the set `indices`
- Improved formulation

```
forall(ind in indices: ind.weight > 150000)
      c1_2: X[ind] >= 1;
```

or

```
dvar int+ X[ind in indices] in
     (ind.weight > 15000 ? 1 : 0)..maxint;
```

Second profiling run with code improvements

Problems | Scripting log | Solutions | Conflicts | Relaxations | Engine log | Statistics | Profiler | CPLEX Servers | Console

Time threshold 20 Memory threshold 20

Description	Time	Time %	Peak Memory	Peak Memory %	Self Time	Self Time %	Local Memory	Local Memory %	Count	Nodes
ROOT	0.3432	100%	11.961 M	100%	0.1092	32%	6.917 M	58%	1	23
READ_DEFINITION sample1	0.0000	0%	0 B	0%	0.0000	0%	128 B	0%	1	1
LOAD_MODEL sample1-16C81A18	0.1404	41%	7.277 M	61%	0.0000	0%	5.949 M	50%	1	12
LOAD_DATA C:\WSTE-examples\c	0.1248	36%	3.492 M	29%	0.0000	0%	3.159 M	26%	1	7
INIT indicies	0.0624	18%	2.473 M	21%	0.0624	18%	1.938 M	16%	1	3
INIT TIndex	0.0000	0%	16 K	0%	0.0000	0%	16.805 K	0%	1	2
INIT TProp	0.0000	0%	16 K	0%	0.0000	0%	392 B	0%	1	1
INIT properties	0.0000	0%	0 B	0%	0.0000	0%	75.102 K	1%	1	1
INIT map	0.0624	18%	1 M	8%	0.0624	18%	1.148 M	10%	1	2
INIT TMap	0.0000	0%	0 B	0%	0.0000	0%	392 B	0%	1	1
PRE_PROCESSING	0.0000	0%	2.031 M	17%	0.0000	0%	1.911 M	16%	1	3
EXECUTE anonymous#1	0.0000	0%	2.031 M	17%	0.0000	0%	1.911 M	16%	1	2
INIT copy_of_indicies	0.0000	0%	2.031 M	17%	0.0000	0%	1.91 M	16%	1	1
INIT X	0.0156	5%	620 K	5%	0.0156	5%	581.18 K	5%	1	1
EXTRACT sample1-16C81A18	0.0780	23%	2.785 M	23%	0.0312	9%	904.129 K	7%	1	6
OBJECTIVE	0.0000	0%	1.398 M	12%	0.0000	0%	78.469 K	1%	1	1
EXTRACT c1_2	0.0312	9%	828 K	7%	0.0312	9%	188 B	0%	2449	2
INIT c1_2	0.0000	0%	0 B	0%	0.0000	0%	40.102 K	0%	1	1
EXTRACT c2	0.0156	5%	592 K	5%	0.0156	5%	362 B	0%	995	2
INIT c2	0.0000	0%	0 B	0%	0.0000	0%	4.227 K	0%	1	1
CPLEX MIP Optimization	0.0156	5%	1.984 M	17%	0.0000	0%	1.473 M	12%	1	2
CPLEX Pre Solve	0.0156	5%	1.223 M	10%	0.0156	5%	1.223 M	10%	1	1
POST_PROCESSING	0.0000	0%	0 B	0%	0.0000	0%	272 B	0%	1	1

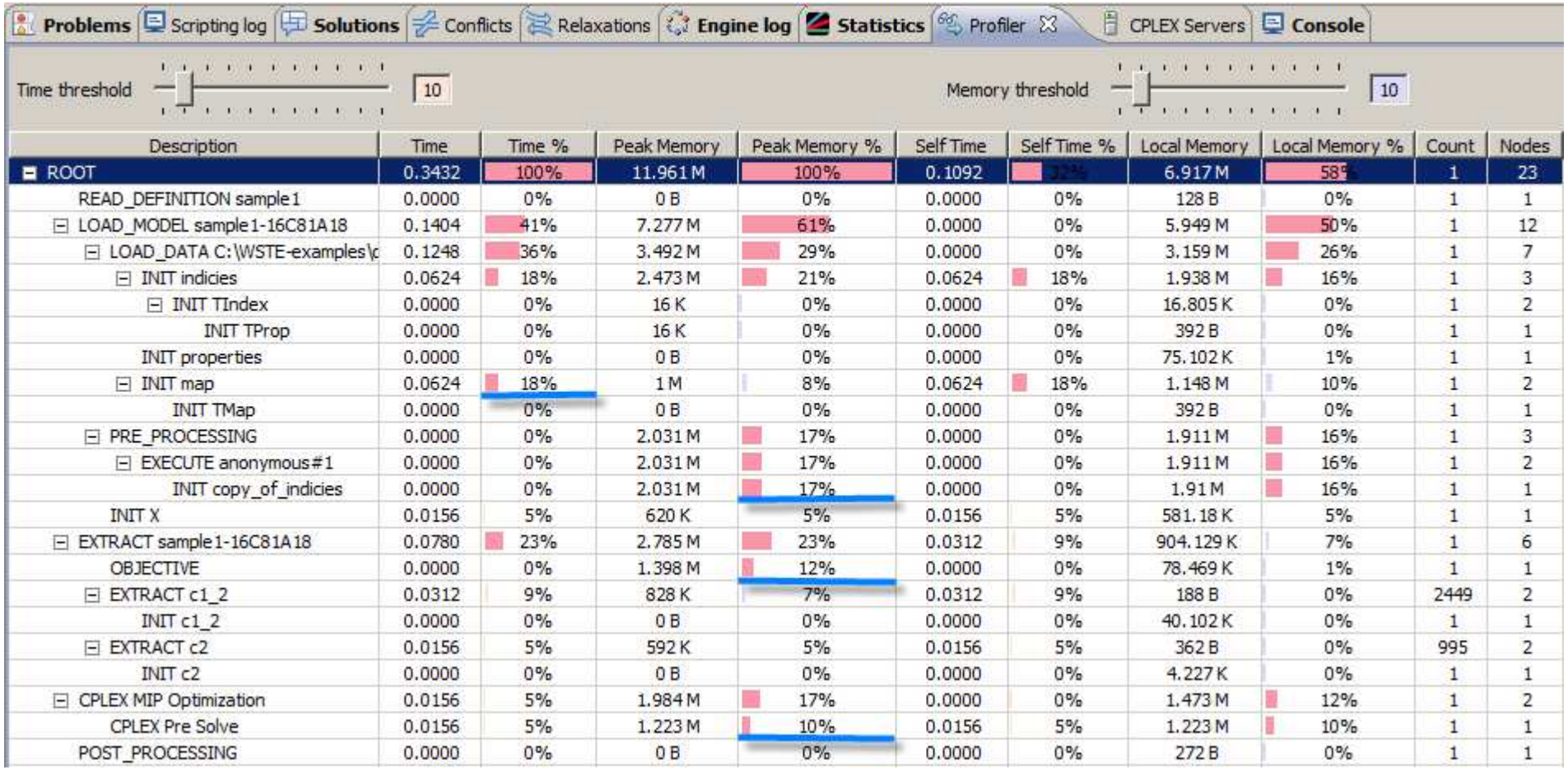
Performance Analysis of “INIT indices” (21% of memory)

- Related Codes

```
tuple TProp {
  string prop;
  int limit;
}

tuple TIndex {
  int i;
  TProp prop1;
  TProp prop2;
  TProp prop3;
  int weight;
};

{TIndex} indices = ...;
```

- Analysis
  - This is plainly data read from data files. Not much to improve.

Reduce threshold to 10%



Description	Time	Time %	Peak Memory	Peak Memory %	Self Time	Self Time %	Local Memory	Local Memory %	Count	Nodes
ROOT	0.3432	100%	11.961 M	100%	0.1092	32%	6.917 M	58%	1	23
READ_DEFINITION sample1	0.0000	0%	0 B	0%	0.0000	0%	128 B	0%	1	1
LOAD_MODEL sample1-16C81A18	0.1404	41%	7.277 M	61%	0.0000	0%	5.949 M	50%	1	12
LOAD_DATA C:\WSTE-examples\c	0.1248	36%	3.492 M	29%	0.0000	0%	3.159 M	26%	1	7
INIT indicies	0.0624	18%	2.473 M	21%	0.0624	18%	1.938 M	16%	1	3
INIT TIndex	0.0000	0%	16 K	0%	0.0000	0%	16.805 K	0%	1	2
INIT TProp	0.0000	0%	16 K	0%	0.0000	0%	392 B	0%	1	1
INIT properties	0.0000	0%	0 B	0%	0.0000	0%	75.102 K	1%	1	1
INIT map	0.0624	18%	1 M	8%	0.0624	18%	1.148 M	10%	1	2
INIT TMap	0.0000	0%	0 B	0%	0.0000	0%	392 B	0%	1	1
PRE_PROCESSING	0.0000	0%	2.031 M	17%	0.0000	0%	1.911 M	16%	1	3
EXECUTE anonymous#1	0.0000	0%	2.031 M	17%	0.0000	0%	1.911 M	16%	1	2
INIT copy_of_indicies	0.0000	0%	2.031 M	17%	0.0000	0%	1.91 M	16%	1	1
INIT X	0.0156	5%	620 K	5%	0.0156	5%	581.18 K	5%	1	1
EXTRACT sample1-16C81A18	0.0780	23%	2.785 M	23%	0.0312	9%	904.129 K	7%	1	6
OBJECTIVE	0.0000	0%	1.398 M	12%	0.0000	0%	78.469 K	1%	1	1
EXTRACT c1_2	0.0312	9%	828 K	7%	0.0312	9%	188 B	0%	2449	2
INIT c1_2	0.0000	0%	0 B	0%	0.0000	0%	40.102 K	0%	1	1
EXTRACT c2	0.0156	5%	592 K	5%	0.0156	5%	362 B	0%	995	2
INIT c2	0.0000	0%	0 B	0%	0.0000	0%	4.227 K	0%	1	1
CPLEX MIP Optimization	0.0156	5%	1.984 M	17%	0.0000	0%	1.473 M	12%	1	2
CPLEX Pre Solve	0.0156	5%	1.223 M	10%	0.0156	5%	1.223 M	10%	1	1
POST_PROCESSING	0.0000	0%	0 B	0%	0.0000	0%	272 B	0%	1	1

Performance Analysis of “INIT map” (18% of time)

- Related Codes

```
{TMap} map with prop in properties, ind in indices= ...;
```

- Analysis
  - Similar to “INIT indices”, not much to improve.
  - Can squeeze out some performance by skipping data consistency check (the `with` keyword)
    - Data consistency might be more important than the performance gain
    - Can implement consistency validation at database
    - New setting datachecks added in version 12.5.1

Performance Analysis of “INIT copy_of_indices” (17% of memory)

- Related Codes

```
{TIndex} copy_of_indices = {ind | ind in indices};

execute {
  copy_of_indices; // simulating temporary data usage
}
```

- Analysis
  - Temporary data can be ended to save memory.
- Improvement

```
execute {
   copy_of_indices.end();
}
```

Performance Analysis of “OBJECTIVE” (12% of memory)

- Related Codes

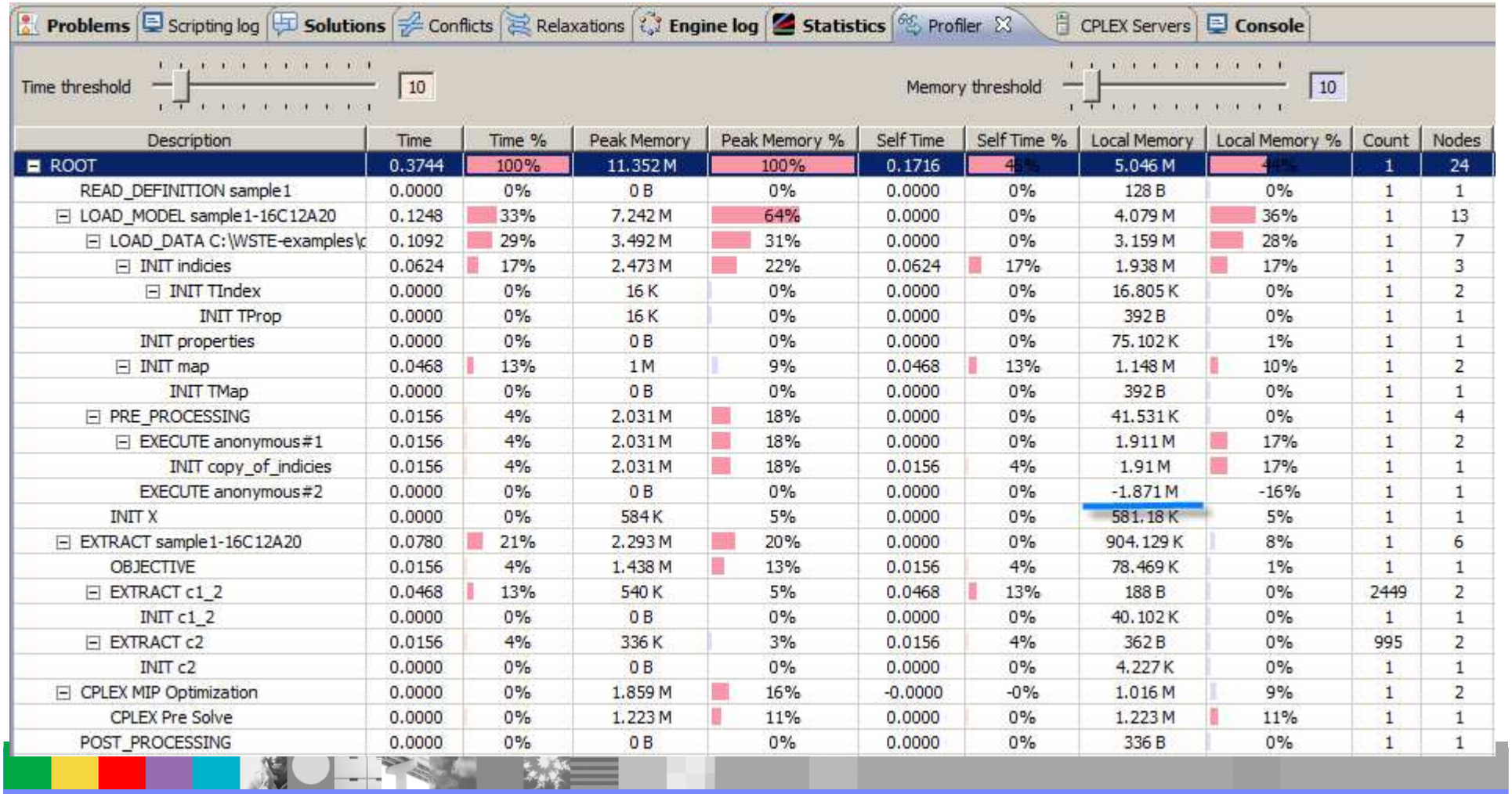
```
dvar int+ X[indices];

minimize sum(ind in indicies) X[ind];
```

- Analysis
  - Nothing to improve, as all of the variables are used in the constraints and objective.
  - However, one may consider a sparse formulation when a significant number of variables are not used in the model.

Performance Analysis of “CPLEX Pre Solve” (10% of memory)

- This number is not related to OPL model generation.
- Presolve is an important feature of CPLEX that is critical for CPLEX’s performance.
- One should not be too concerned by its profiling results unless the overall performance of optimization engines (CPLEX Optimizer or CP Optimizer) looks bad.

Third profiling run with further code improvements (ending temp data)



Description	Time	Time %	Peak Memory	Peak Memory %	Self Time	Self Time %	Local Memory	Local Memory %	Count	Nodes
ROOT	0.3744	100%	11.352 M	100%	0.1716	4	5.046 M	4	1	24
READ_DEFINITION sample1	0.0000	0%	0 B	0%	0.0000	0%	128 B	0%	1	1
LOAD_MODEL sample1-16C12A20	0.1248	33%	7.242 M	64%	0.0000	0%	4.079 M	36%	1	13
LOAD_DATA C:\WSTE-examples\c	0.1092	29%	3.492 M	31%	0.0000	0%	3.159 M	28%	1	7
INIT indicies	0.0624	17%	2.473 M	22%	0.0624	17%	1.938 M	17%	1	3
INIT TIndex	0.0000	0%	16 K	0%	0.0000	0%	16.805 K	0%	1	2
INIT TProp	0.0000	0%	16 K	0%	0.0000	0%	392 B	0%	1	1
INIT properties	0.0000	0%	0 B	0%	0.0000	0%	75.102 K	1%	1	1
INIT map	0.0468	13%	1 M	9%	0.0468	13%	1.148 M	10%	1	2
INIT TMap	0.0000	0%	0 B	0%	0.0000	0%	392 B	0%	1	1
PRE_PROCESSING	0.0156	4%	2.031 M	18%	0.0000	0%	41.531 K	0%	1	4
EXECUTE anonymous#1	0.0156	4%	2.031 M	18%	0.0000	0%	1.911 M	17%	1	2
INIT copy_of_indicies	0.0156	4%	2.031 M	18%	0.0156	4%	1.91 M	17%	1	1
EXECUTE anonymous#2	0.0000	0%	0 B	0%	0.0000	0%	-1.871 M	-16%	1	1
INIT X	0.0000	0%	584 K	5%	0.0000	0%	581.18 K	5%	1	1
EXTRACT sample1-16C12A20	0.0780	21%	2.293 M	20%	0.0000	0%	904.129 K	8%	1	6
OBJECTIVE	0.0156	4%	1.438 M	13%	0.0156	4%	78.469 K	1%	1	1
EXTRACT c1_2	0.0468	13%	540 K	5%	0.0468	13%	188 B	0%	2449	2
INIT c1_2	0.0000	0%	0 B	0%	0.0000	0%	40.102 K	0%	1	1
EXTRACT c2	0.0156	4%	336 K	3%	0.0156	4%	362 B	0%	995	2
INIT c2	0.0000	0%	0 B	0%	0.0000	0%	4.227 K	0%	1	1
CPLEX MIP Optimization	0.0000	0%	1.859 M	16%	-0.0000	-0%	1.016 M	9%	1	2
CPLEX Pre Solve	0.0000	0%	1.223 M	11%	0.0000	0%	1.223 M	11%	1	1
POST_PROCESSING	0.0000	0%	0 B	0%	0.0000	0%	336 B	0%	1	1

Not Designed For…

- The numbers reported by OPL profiler are not very precise.
  - Not accurate for byte-level, or millisecond-level analysis
  - The bigger the numbers, the more accurate they are.
- This methodology is more for “tactical” issues than “strategic” issues.
  - It detects possible inefficiencies in implementation.
  - It does not always give sufficient hints to “strategic” problems, for example:
    - If an element of a tuple is not referenced at all
    - If the model is symmetric, but model has not taken full advantages of it.
    - If a multi-stage model can be decomposed into smaller models (one model for each stage), and the time to solve all of the smaller models is faster than solving the original model.

Other Suggestions

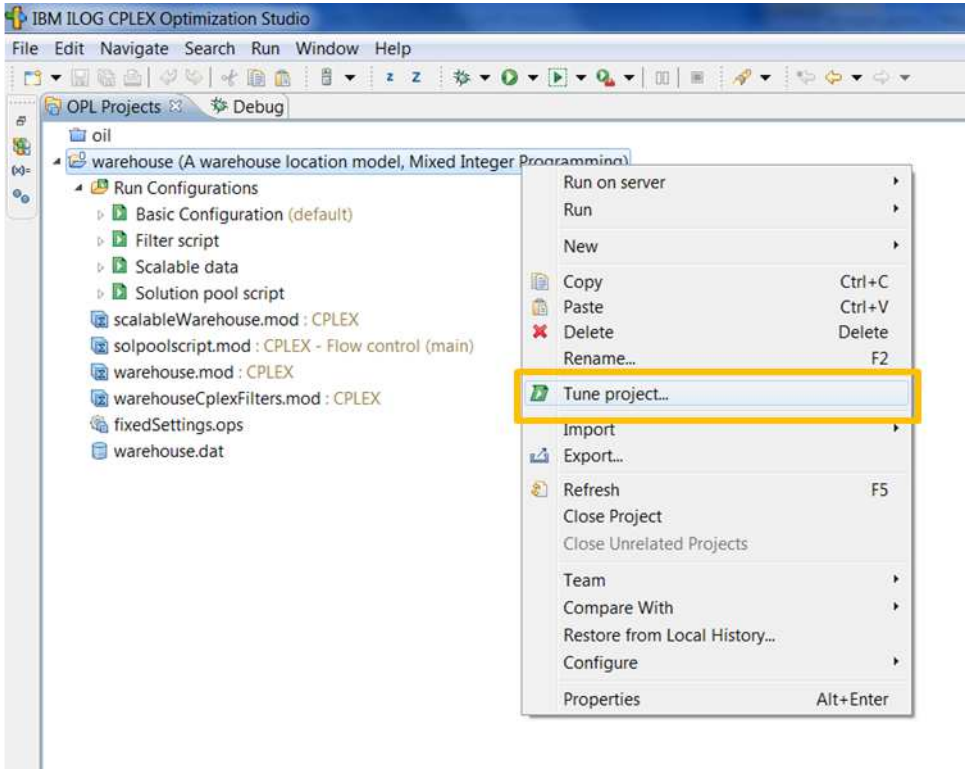
- Profile models with realistic data sets to understand whether the models would scale well in the target deployment environments.
- Use multiple data sets. It is possible that one data set only stresses part of the model.
  - The sample batch can be extended to automate the runs in one batch.
- Do profiling periodically:
  - Don’t do it too late. Otherwise, you might need to rewrite a major part of your model to achieve the desired level of improvements.

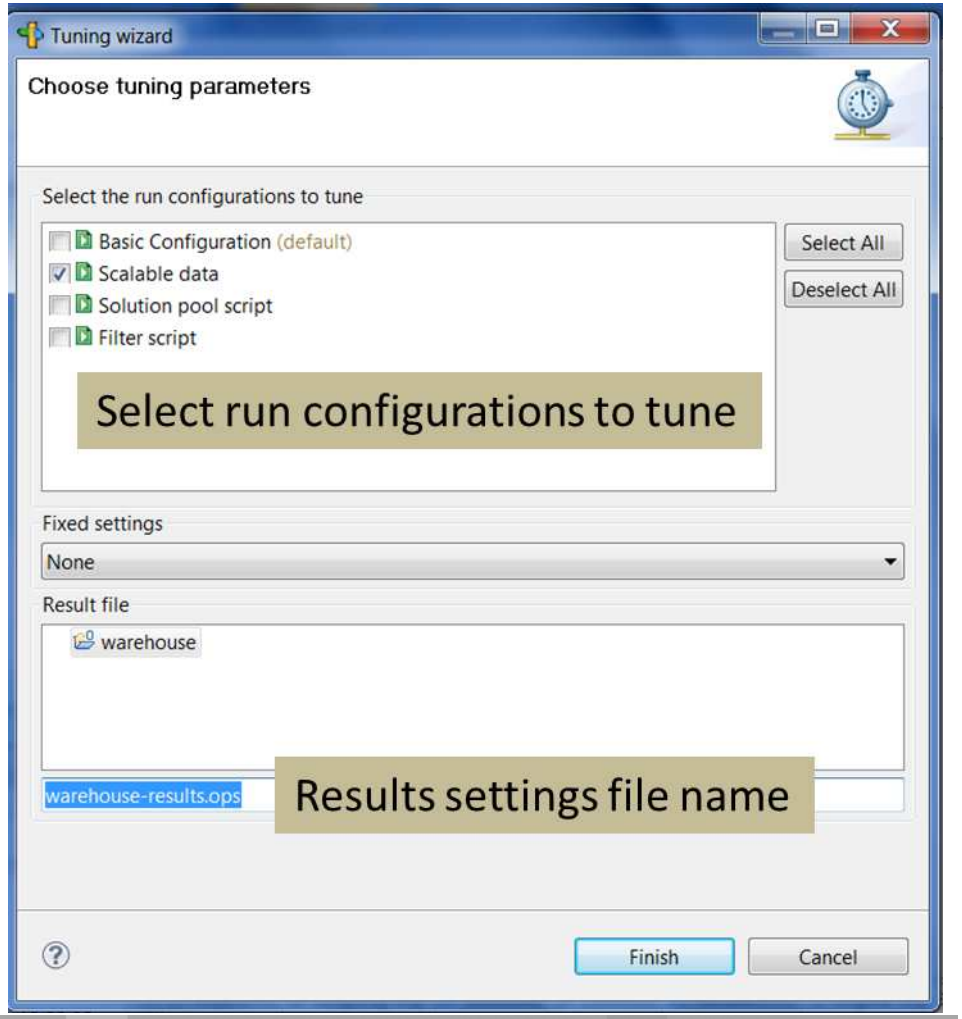
Performance Tuning using the Tuning Tool

What is the Tuning Tool

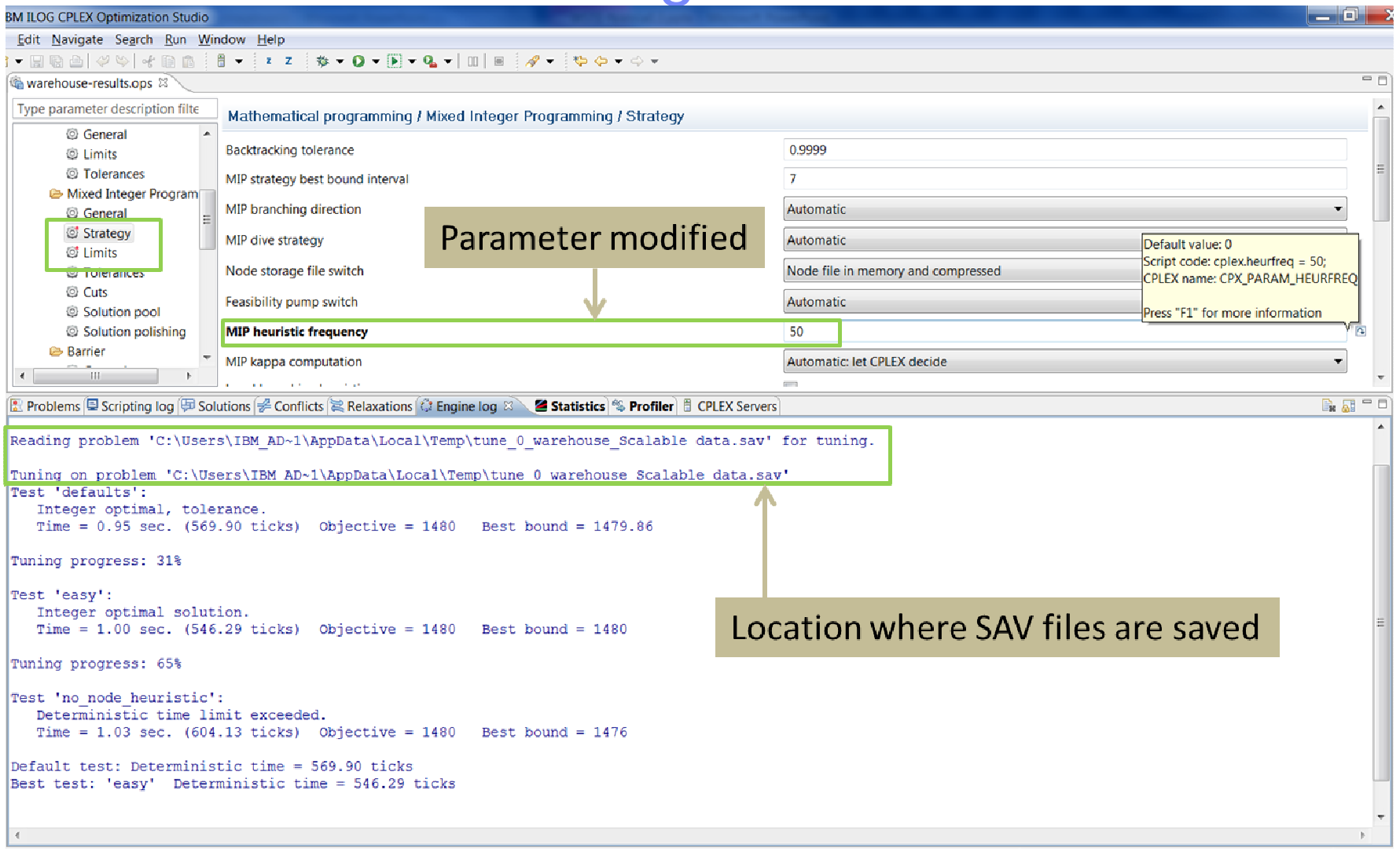
- Perform some baseline performance tuning
  - Based on the “tune” feature of CPLEX
  - Option to tune with a fixed set of CPLEX parameters
  - Can be run over one or several run configurations – helps in tuning on a set of problem instances
    - Tune for average performance or provide the least worst performance for a set of models
  - Produce SAV files of the models, which can be later used to analyze using the CPLEX Interactive Optimizer
- Tuning tool is not yet available for CP Optimizer

How to invoke the Tuning Tool

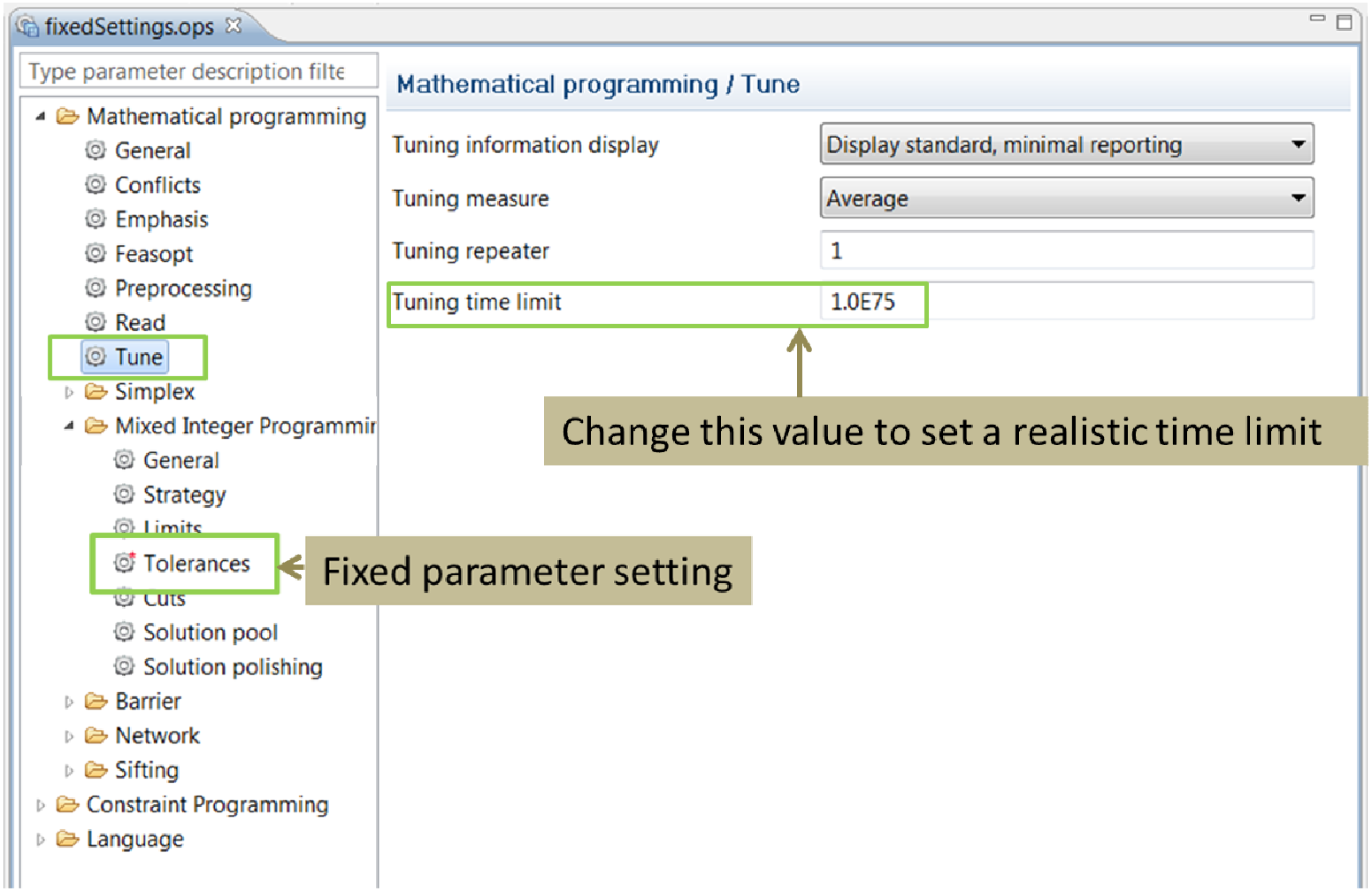




Results from the Tuning Tool

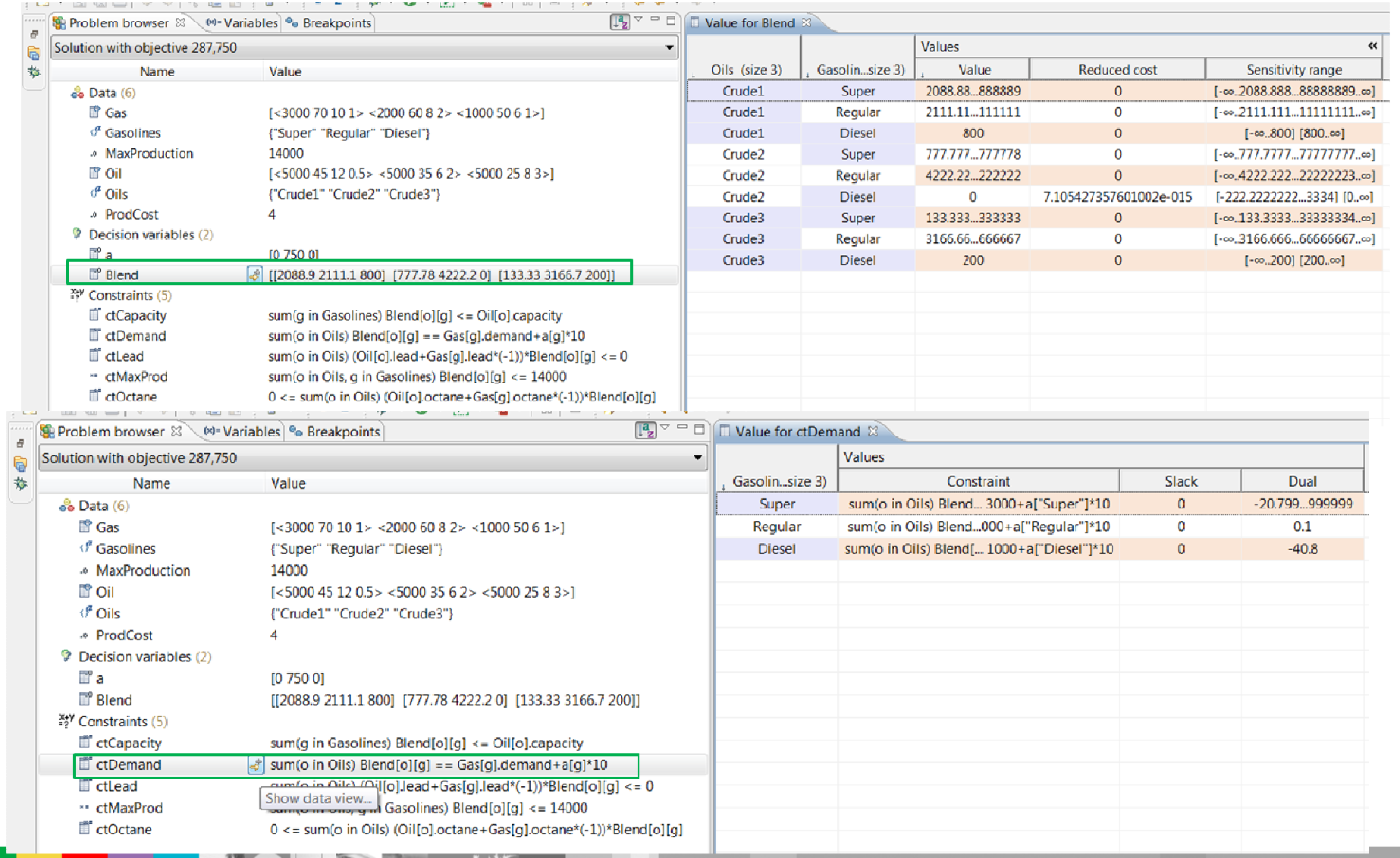


Customizing the Tuning Tool

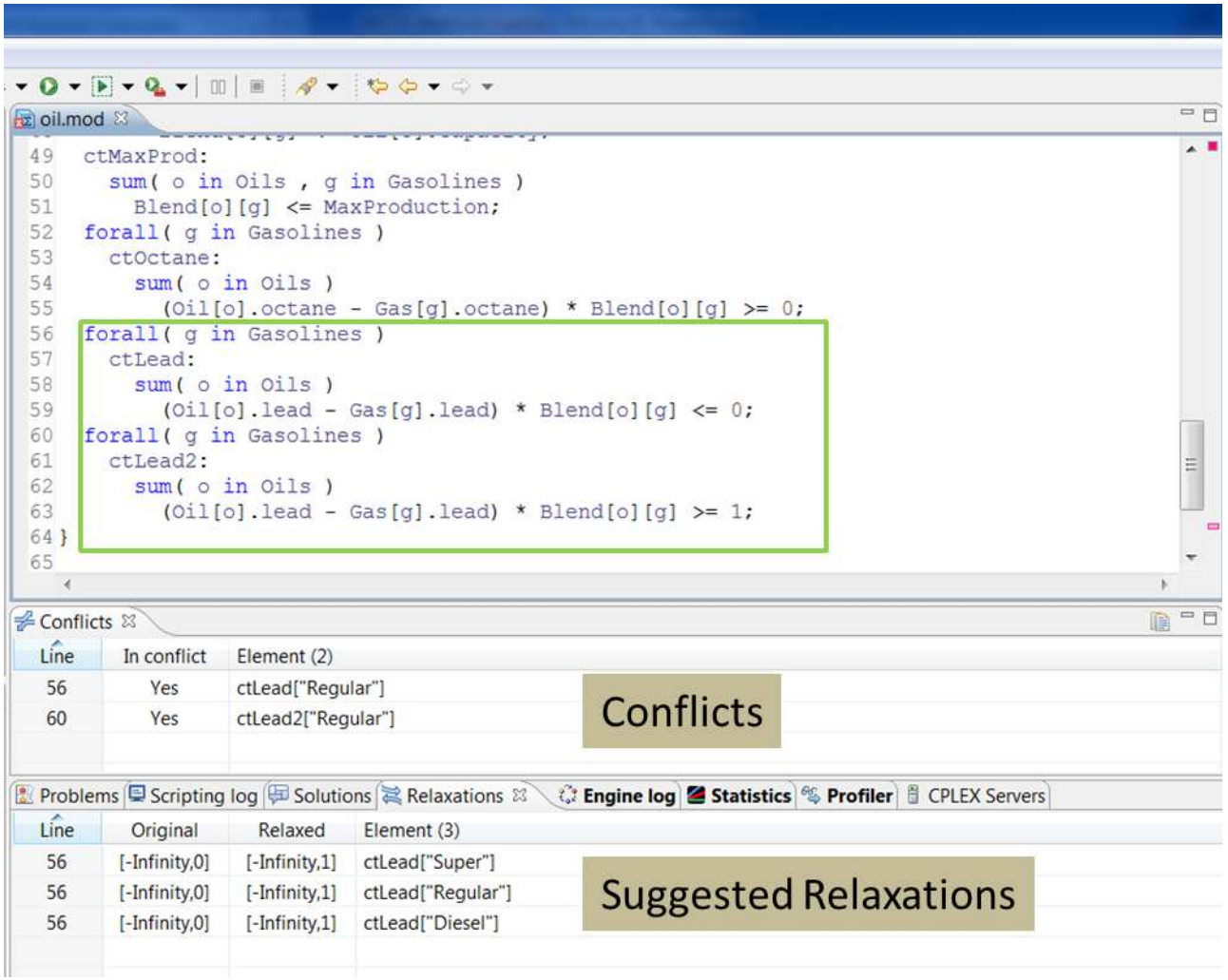


Some Other Features of the IDE

Problem Browser



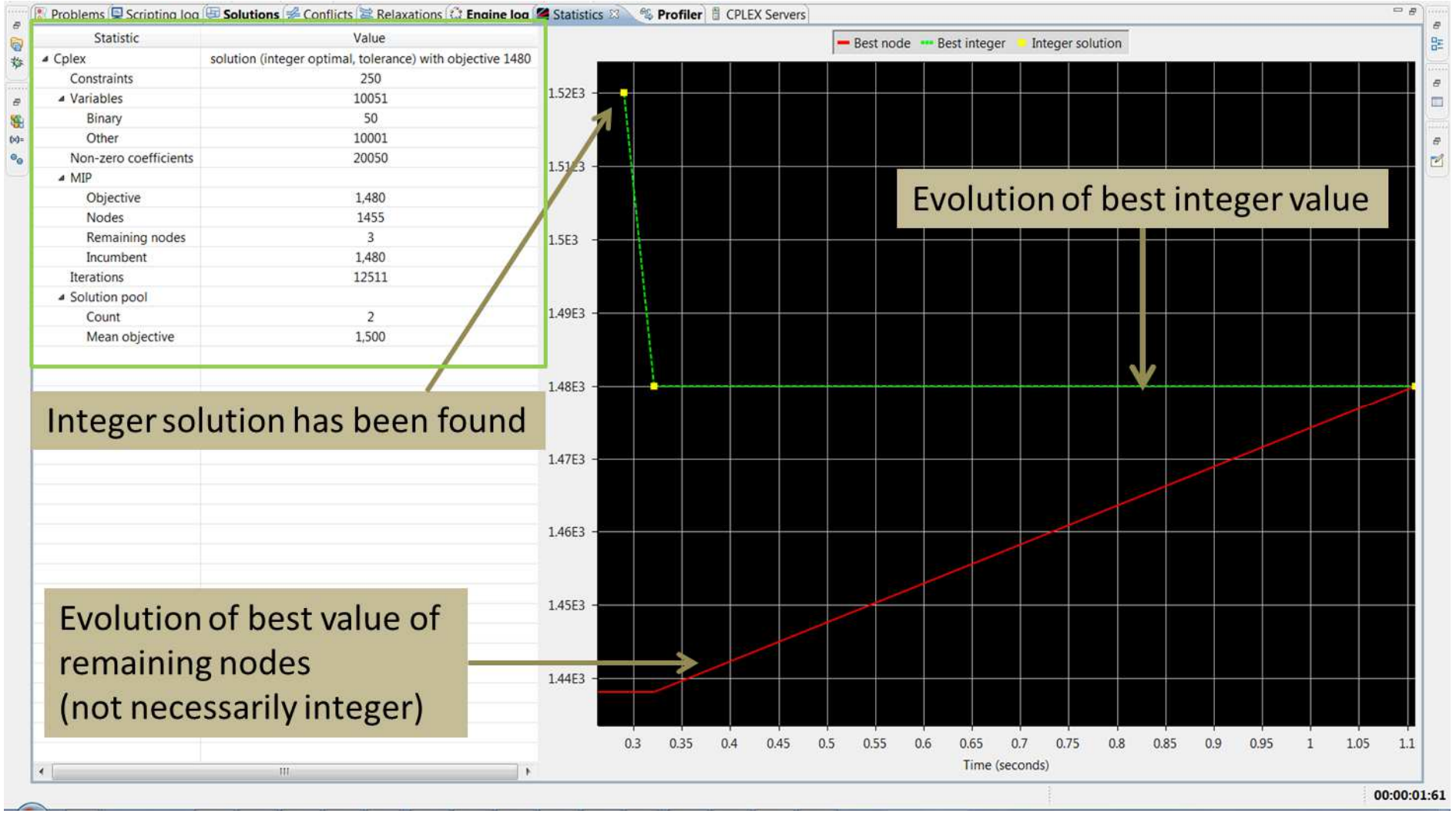
Conflicts and Relaxations



Conflicts and Relaxations

- You can select the relaxation levels
  - Only labeled constraints
  - All variables and labeled constraints (default)
- Un-labeled constraints are not considered for conflict and relaxation search process – remember to label constraints

Statistics and Progress Chart (MIP)



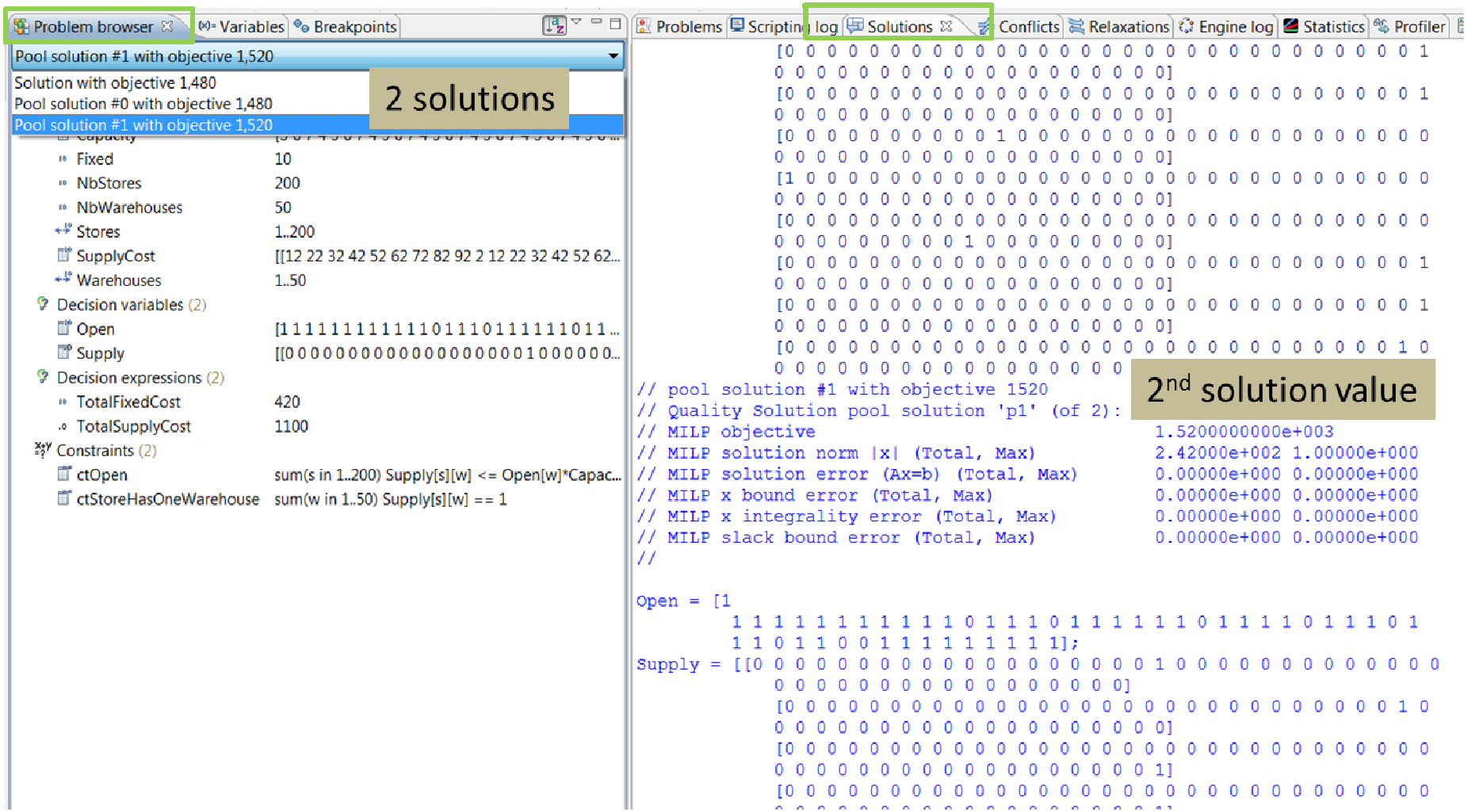
Solution Pool in the IDE

- Helps to generate and store multiple solutions for a Mixed Integer Programming (MIP) model
  - Sometimes solutions with slightly worse objective values are more attractive than the mathematically optimal solution
  - Or sometimes one might want to examine alternate optimal solutions
- Two ways to store these solutions
  - Default mode – Store any feasible solution found
  - Populate – Called after MIP search to find more solutions
- Solution Pool filters
  - Diversity Filters – generate solutions similar to (or different) from a set of reference values
  - Range Filters – generate solutions that obey a new constraint, specified as a linear expression within a range

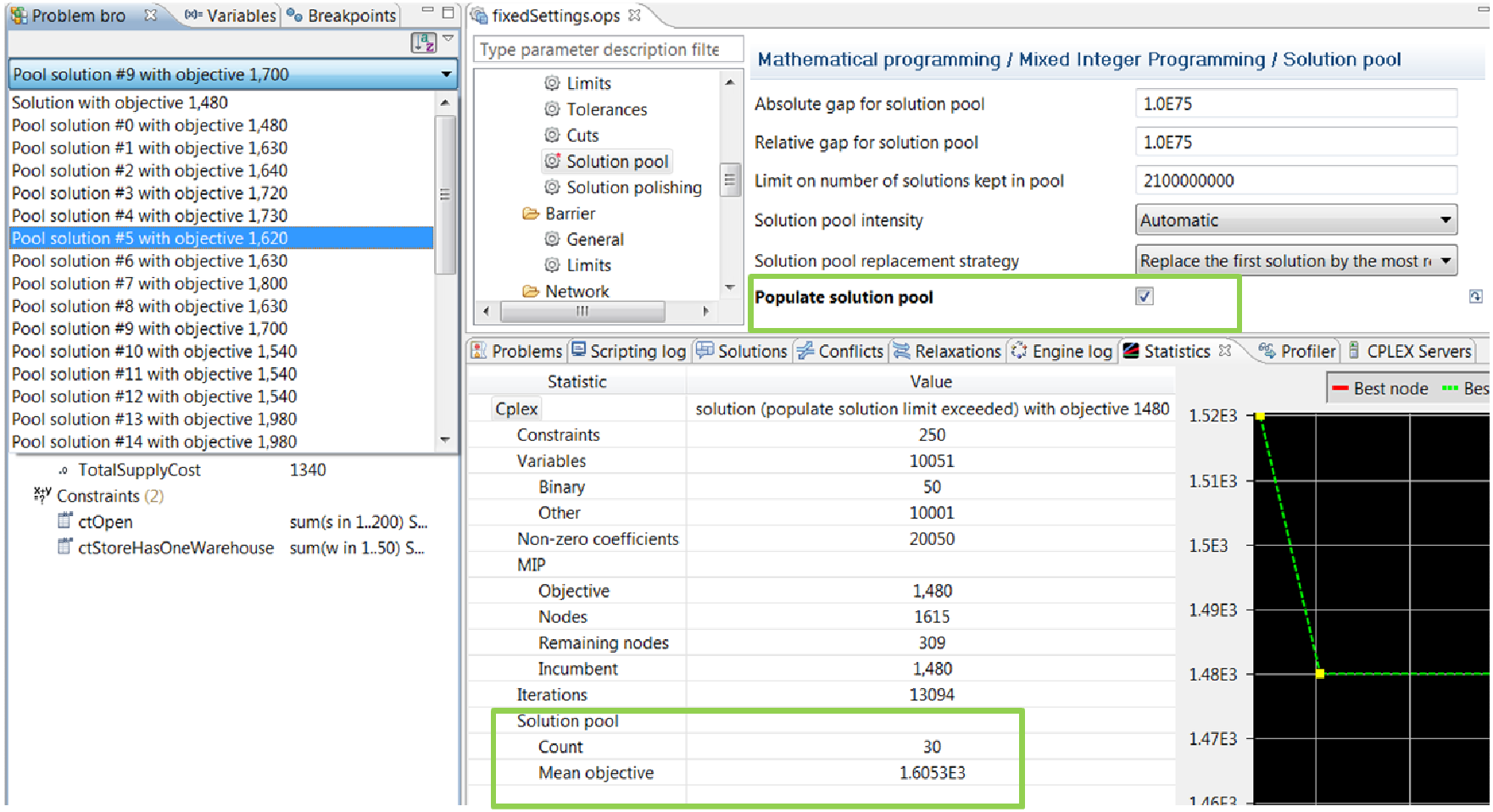
Where can I find the Solution Pool in the IDE?

- Problem Browser displays the different solutions in the Solution Pool
- Statistics tab provides the number of solutions found and the mean objective value
- When you select solution from Problem Browser, values are displayed in Solutions tab

Solution Pool in the IDE



Activating Populate Solution Pool



Installing CPLEX Optimization Studio on Windows

Few points to consider before Installing

- On Windows 7 and Windows Vista machines, run the installer by right clicking on the executable and use the option Run as Administrator
- For better compatibility with your environment, install in a non-Program files location, for example C:\ILOG
- Check Path environment variable – should not have any non-ASCII characters
- Ensure Security Software (Antivirus, Firewall) is not blocking the installer from running
- You will require Visual C++ Runtime Libraries to use the IBM ILOG CPLEX Optimization Studio IDE

Summary

- We introduced a methodology to use OPL Profiler to fine tune the performance of model generation
  - The reformulation techniques are also important
- We introduced the OPL tune feature to perform some baseline performance tuning
- CPLEX Optimization Studio also offers some other features that help users work on their models

Technotes and Documentation Resources

- The white paper describing efficient OPL model development
  ftp://public.dhe.ibm.com/common/ssi/rep_wh/n/WSW14059USEN/WSW14059USEN.PDF
- Documentation of the OPL Profiler:
  http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5/topic/ilog.odms.ide.help/OPL_Studio/usroplide/topics/opl_ide_profil.html
- Documentation to the OPL Tuning tool:
  http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r5/topic/ilog.odms.ide.help/OPL_Studio/usroplide/topics/opl_ide_perf.html
- Links to the CPLEX Optimization Studio Information Center
  http://www-01.ibm.com/support/docview.wss?uid=swg21503602

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community:
  http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe
2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com
3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Questions and Answers

This Support Technical Exchange session will be recorded and a replay will be available on IBM.COM sites and possibly social media sites such as YouTube. When speaking, **do not state any confidential information, your name, company name or any information you do not want shared publicly in the replay**. By speaking in during this presentation, you assume liability for your comments.

Supplement Information for Slide 11

OPL Codes Created

The files

- We created the files from scratch based on real customer requests.
- The common tuple definitiond and the main model has been fully shown on slide 11.
- The next slide contains the data generation codes.

```
include "common.mod";

{TIndex} indices;

{TProp} properties;

{TMap} map;

execute {

  for(var a = 0; a < 1000; a++){
    var temp = 100000 + Opl.rand(100000);
    properties.add("pp_"+temp.toString(), temp);
  }

  var properties_size = properties.size;

  for(var a = 0; a < 10000; a++){
    indices.add(a,
                Opl.item(properties,Opl.rand(properties_size)),
                Opl.item(properties,Opl.rand(properties_size)),
                Opl.item(properties,Opl.rand(properties_size)),
                Opl.rand(200000));
  }

  var indices_size = indices.size;

  for(var a = 0; a < 4000; a++){
    map.add(Opl.item(indices,Opl.rand(indices_size)),
                Opl.item(properties,Opl.rand(properties_size)));
  }
}

execute {

  var f = new IloOplOutputFile("data.dat");
  f.writeln("indices = " + indices + ";");
  f.writeln("properties = " + properties + ";");
  f.writeln("map = " + map + ";");
  f.close();
}
```

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM’S CURRENT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION, NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO NOR SHALL HAVE THE EFFECT OF CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCT OR SOFTWARE.

51