CMPL
<Coin|Coliop> Mathematical Programming Language

Mike Steglich

Technical University of Applied Sciences Wildau

ICMS 2016

The 5th International Congress on Mathematical Software

Berlin

13 July 2016

1 Introduction

- **CMPL** (<Coliop|Coin> Mathematical Programming Language) is a mathematical programming language and a system for mathematical programming and optimisation of linear optimisation problems. CMPL executes CBC (default), CPLEX, GLPK, Gurobi and SCIP directly to solve the generated model instance.

- The CMPL distribution contains **Coliop** which is an IDE (Integrated Development Environment) for CMPL.
- **pyCMPL** is the CMPL application programming interface (API) for Python and an interactive shell and **jCMPL** is CMPL's Java API.
- **CMPLServer** is a XML-RPC-based web service for distributed and grid optimisation that can be used with CMPL, pyCMPL and jCMPL.

- The CMPL distribution is an **Open Source Project** and available for most of the relevant operating systems (Windows, OS X, Linux and Raspbian).
- CMPL, Coliop, pyCMPL, jCMPL and CMPLServer are **COIN-OR** projects initiated by the **Technical University of Applied Sciences Wildau** and the Institute for Operations Research and Business Management at the Martin Luther University Halle-Wittenberg.

- **CMPL** and **pyCMPL** are also available in **SolverStudio**, which is an add-in for Excel 2007 and later on Windows created by Andrew Mason (University Auckland, NZ) that allows a user to build and solve optimisation models in Excel optimisation modelling languages.

2 Selected language elements and features

2.1 Basic language elements

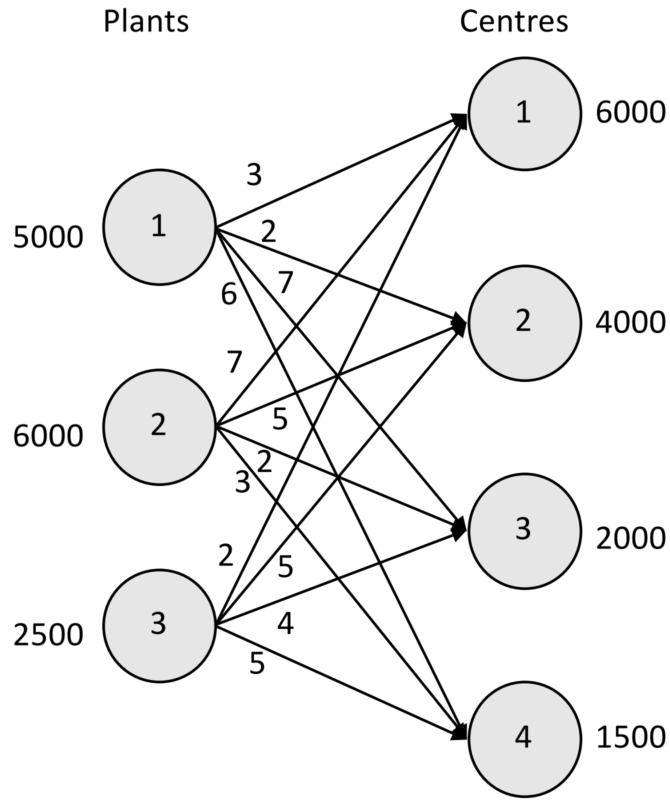
$$\sum_{i \in S} \sum_{j \in D} c_{ij} \cdot x_{ij} \rightarrow \min!$$

$$s.t.$$

$$\sum_{j \in D} x_{ij} = s_i \qquad ; i \in S$$

$$\sum_{i \in S} x_{ij} = d_j \qquad ; j \in D$$

$$x_{ij} \geq 0 \qquad ; i \in S, j \in D$$



[Anderson et al. (2014), p. 281.]

Parameters*:*

- *S* set of the sources
- *D* set of the destinations
- s_i supply in units of source *i*
- d_j demand in units of destination *j*
- c_{ij} transportation costs per unit if a unit is shipped from source *i* to the destination *j*

Variables:

- x_{ij} units that are to be transported from source *i* to the destination *j*

[cf. Hillier and Liebermann (2010), p. 305ff, Williams (2013), p. 82ff, Vanderbei (2014), p. 225ff.]

2.1 Basic language elements

$$\sum_{i\in S}\sum_{j\in D} c_{ij}\cdot x_{ij} \rightarrow \min!$$

$$s.t.$$

$$\sum_{j\in D} x_{ij} = s_i \qquad ;i\in S$$

$$\sum_{i\in S} x_{ij} = d_j \qquad ;j\in D$$

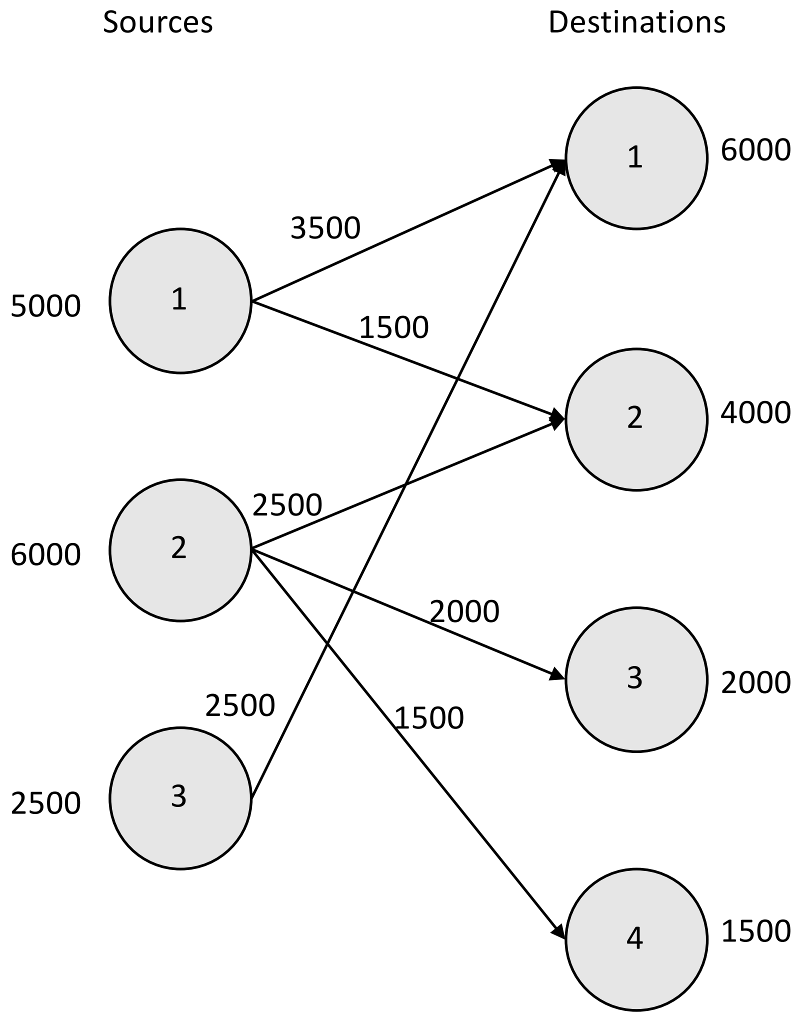
$$x_{ij} \geq 0 \qquad ;i\in S, j\in D$$

```
parameters:
        S:=1..3;
        D:=1..4;
        c[S,D] := ( (3,2,7,6), (7,5,2,3),(2,5,4,5) );
        s[S] := (5000,6000,2500);
        d[D] := (6000,4000,2000,1500);

variables:
        x[S,D]: real[0..];

objectives:
        sum {i in S, j in D: c[i,j] * x[i,j] } ->min;

constraints:
        {i in S: sum{j in D: x[i,j]} = s[i];}
        {j in D: sum{i in S: x[i,j]} = d[j];}
```

2.1 Basic language elements

> **cmpl** transportation.cmpl

```
---------------------------------------------------------------------------------------------------------
Problem             transportation.cmpl
Nr. of variables    12
Nr. of constraints  7
Objective name      costs
Solver name         CBC
Display variables   nonzero variables (all)
Display constraints nonzero constraints (all)
---------------------------------------------------------------------------------------------------------

Objective status    optimal
Objective value     39500 (min!)

Variables
Name                Type          Activity       Lower bound       Upper bound          Marginal
---------------------------------------------------------------------------------------------------------
x[1,1]                 C              3500                 0          Infinity                 0
x[1,2]                 C              1500                 0          Infinity                 0
x[2,2]                 C              2500                 0          Infinity                 0
x[2,3]                 C              2000                 0          Infinity                 0
x[2,4]                 C              1500                 0          Infinity                 0
x[3,1]                 C              2500                 0          Infinity                 0
---------------------------------------------------------------------------------------------------------

Constraints
Name                Type          Activity       Lower bound       Upper bound          Marginal
---------------------------------------------------------------------------------------------------------
supply[1]              E              5000              5000              5000                 1
supply[2]              E              6000              6000              6000                 4
supply[3]              E              2500              2500              2500                 -
demand[1]              E              6000              6000              6000                 2
demand[2]              E              4000              4000              4000                 1
demand[3]              E              2000              2000              2000                -2
demand[4]              E              1500              1500              1500                -1
---------------------------------------------------------------------------------------------------------
```



[Anderson et al. (2014), p. 284.]

2.2 CMPL header and CmplData

- **CMPL header**
  - A CMPL header is intended to define CMPL options, solver options and display options for the specific CMPL model. An additional intention of the CMPL header is to specify external data files which are to be connected to the CMPL model. The elements of the CMPL header are not part of the CMPL model and are processed before the CMPL model is interpreted.
- **CmplData**
  - A CmplData file is a plain text file that contains the definition of parameters and sets with their values in a specific syntax. The parameters and sets can be read into a CMPL model by using the CMPL header argument %data.

`%arg -solver glpk`
`%arg -cmplUrl ↲`
`http://194.95.44.187:8008`
	GLPK is to be executed on a CmplServer located at 194.95.44.187:8008

`%opt cbc ratio 0.1`	If CBC is chosen then a mipGap of 0.1 is used
`%display var x*`	Only variables starting with `x` are to be displayed in the solution report.

2.2 CMPL header and CmplData

transportation.cdat

```
%S set <1..3>
%D set <1..4>
%s[S] < 5000 6000 2500 >
%d[D] < 6000 4000 2000 1500 >
%c[S, D] <     3 2 7 6
               7 5 2 3  2 5 4 5 >
```

transportation.cmpl

```
%data transportation.cdat : S set, D set, c[S,D], s[S], d[D]
%display nonZeros

variables:
        x[S,D]: real[0..];
objectives:
        sum {i in S, j in D: c[i,j] * x[i,j] } ->min;
constraints:
        {i in S: sum{j in D: x[i,j]} = s[i];}
        {j in D: sum{i in S: x[i,j]} = d[j];}
```

2.3 List of the language elements

- Objects
  - Parameters
  - Variables
  - Indices and sets
  - Line names
- CMPL header
- Parameter Expressions
  - Array functions
  - Set operations and functions
  - Mathematical functions
  - Type casts
  - String operations
- Input and output operations
  - Error and user messages
  - cmplData files
  - Readcsv and readstdin
  - Include
- Control structures
  - For loop
  - If-then clause
  - Switch clause
  - While loop
  - Set and sum control structures as expression
- Matrix-Vector notations
- Automatic model reformulations
  - Matrix reductions
  - Equivalent transformations of products of variables

3 Application programming interfaces (pyCmpl and jCmpl)

- **pyCMPL** is the CMPL API for Python and an interactive shell and **jCMPL** is CMPL's Java API.
  - The main idea of the APIs are:
    - to define sets and parameters within the user application,
    - to start and control the solving process and
    - to read the solution(s) into the application if the problem is feasible.
  - All variables, objective functions and constraints are defined in CMPL.
  - These functionalities can be used with a local CMPL installation or a CMPLServer.

3 Application programming interfaces (pyCMPL and jCMPL)

Cost per machine and location

		Locations			
		1	2	3	4
Machines	1	13	16	12	11
	2	15	-	13	20
	3	5	7	10	6

(Hillier/Liebermann 2010, p. 334f.)

$$\sum_{(i,j)\in A} c_{ij}\cdot x_{ij} \to \min!$$

$s.t.$

$$\sum_{\{j:(i,j)\in A\}} x_{ij} = 1 \qquad ;i\in N_1$$

$$\sum_{\{i:(i,j)\in A\}} x_{ij} \le 1 \qquad ;j\in N_2$$

$$x_{ij} \ge 0 \qquad ;(i,j)\in A$$

Parameters:

- N_1 Set of the machines
- N_2 Set of the locations
- A Set of possible assignments
- c_{ij} Assignment costs

Variables:

- x_{ij} Assignment variables

assignment.cmpl

```
%data : machines set, locations set, A set[2], c[A]

variables:
    x[A]: real[0..];

objectives:
    sum{ [i,j] in A : c[i,j]*x[i,j] } -> min ;

constraints:
    { i in machines:  sum{ j in (A *> [i,*]) : x[i,j] } = 1; }
    { j in locations: sum{ i in (A *> [*,j]) : x[i,j] } <= 1;}
```

3 Application programming interfaces (pyCMPL and jCMPL)

- Creating Cmpl object and defining sets and parameters (pyCMPL)

```
m = Cmpl("assignment.cmpl")

locations = CmplSet("locations")
locations.setValues(1,4)

machines = CmplSet("machines")
machines.setValues(1,3)

combinations = CmplSet("A", 2)
combinations.setValues([ [1,1],[1,2],[1,3], ... ])

c = CmplParameter("c",combinations)
c.setValues([13,16,12,11,15,13,20,5,7,10,6])

m.setSets(machines,locations,combinations)
m.setParameters(c)
```

- Solving and manipulating models (pyCMPL)

```
m.setOption("%arg -solver gurobi")

m.solve()
```

3 Application programming interfaces (pyCMPL and jCMPL)

- Reading optimal solution

```
print m.solution.value
print m.solution.status

for v in m.solution.variables:
    print v.name, v.type, v.activity, v.lowerBound, v.upperBound

for c in m.solution.constraints:
    print c.name, c.type, c.activity, c.lowerBound, c.upperBound
```

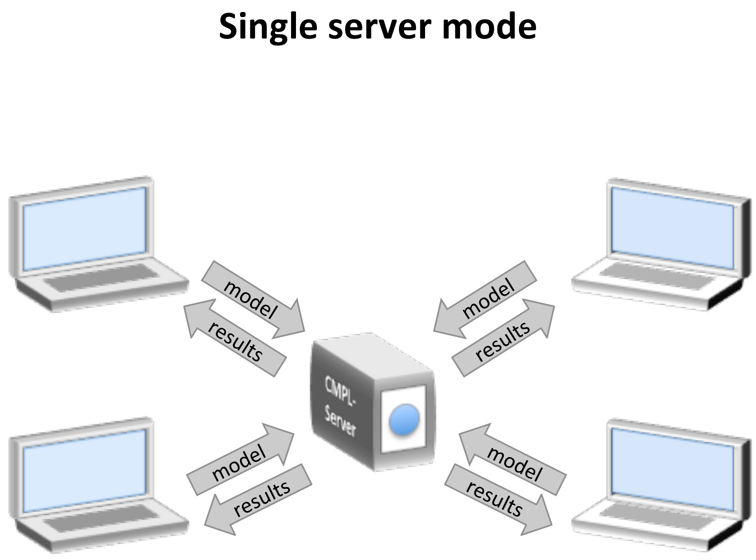
3 Application programming interfaces (pyCMPL and jCMPL)

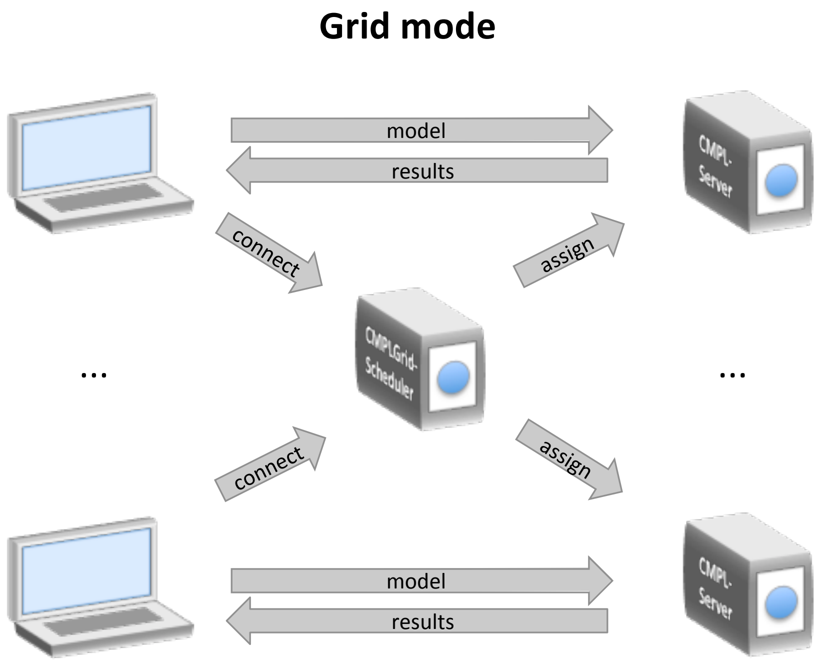
assignment.py

```
#!/usr/bin/python

from pyCmpl import *

try:
     m = Cmpl("assignment.cmpl")

     locations = CmplSet("locations")
     locations.setValues(1,4)

     machines = CmplSet("machines")
     machines.setValues(1,3)

     combinations = CmplSet("A", 2)
     combinations.setValues([[1,1],[1,2],[1,3],[1,4],[2,1],[2,3],[2,4],[3,1],[3,2],[3,3],[3,4]])

     c = CmplParameter("c",combinations)
     c.setValues([13,16,12,11,15,13,20,5,7,10,6])

     m.setSets(machines,locations,combinations)
     m.setParameters(c)

     m.setOption("%arg -solver gurobi")
     m.solve()

     print m.solution.value
     print m.solution.status

     for v in m.solution.variables:
          print v.name, v.type, v.activity, v.lowerBound, v.upperBound

     for c in m.solution.constraints:
          print c.name, c.type, c.activity, c.lowerBound,c.upperBound

except CmplException, e:
     print e.msg
```

4 CMPLServer

- The **CMPLServer** is an **XML-RPC-based web service** (cf. St. Laurent et al. 2001, p. 1.) for distributed and grid optimisation with clients for CMPL, pyCMPL and jCMPL that can be used in two modes:

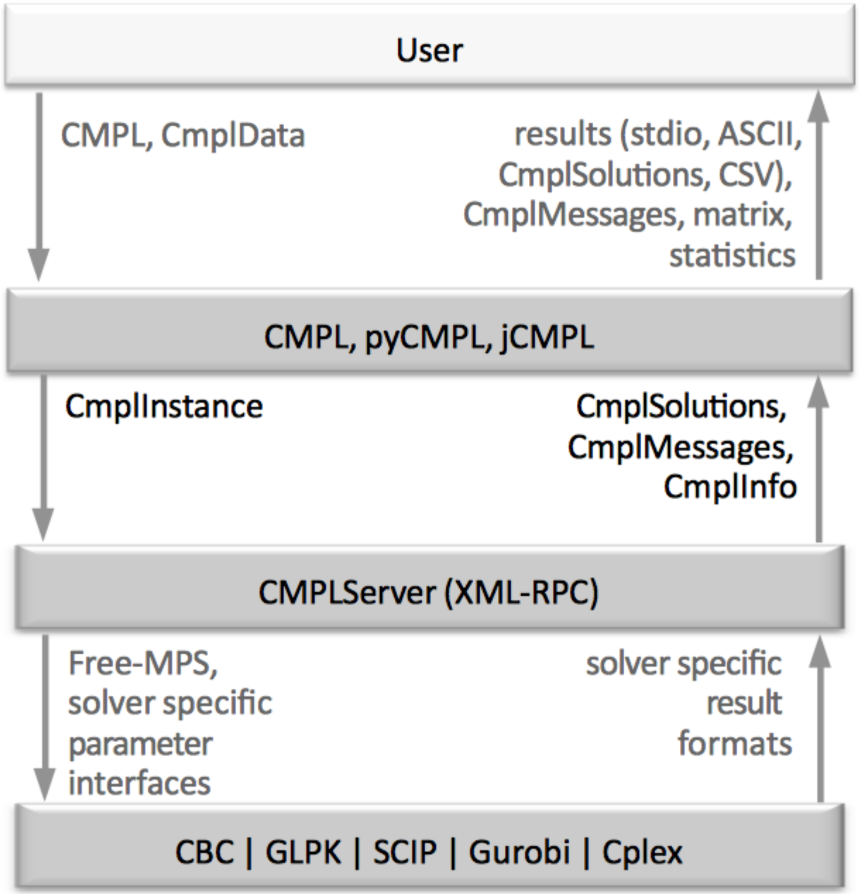




- Both modes can be understood as **distributed systems** “in which hardware and software components located at networks computers communicate and coordinate their actions only by passing messages”. (Coulouris et al, 2012, p. 17)
- **Distributed optimisation** is in this meaning interpretable as a distributed system that can be used for solving optimisation problems. (cf. Kshemkalyani & Singhal, 2008, p. 1; Fourer et.al., 2010)

4.1 Introduction

- The communication between the clients and the server(s) works through **XML-RPC** and four **CMPL-specific XML formats** for the communication between clients and servers.



- **CmplInstance**
  XML file that contains all relevant information about a CMPL model sent to a CMPLServer
- **CmplMessages**
  XML file that contains the status and messages of a CMPL model
- **CmplSolutions**
  CMPL's XML based solution file format
- **CmplInfo**
  XML file that contains (if requested) several statistics and the generated matrix of the CMPL model

(XSD schemes: coliop.org/schemes)

4.2 Single server mode

```
cmplServer –start        # CMPLServer IP 10.0.1.52 port 8008
```

- **CMPL synchronously**

```
cmpl test.cmpl –cmplUrl http://10.0.1.52:8008
```

- **pyCMPL asynchronously**

```
…
m = Cmpl("test.cmpl")
m.connect("http://10.0.1.52:8008")
m.send()
m.knock()
m.retrieve()
…
```

- **jCMPL synchronously**

```
…
m = Cmpl("test.cmpl");
m.connect("http://10.0.1.52:8008");
m.solve();
…
```

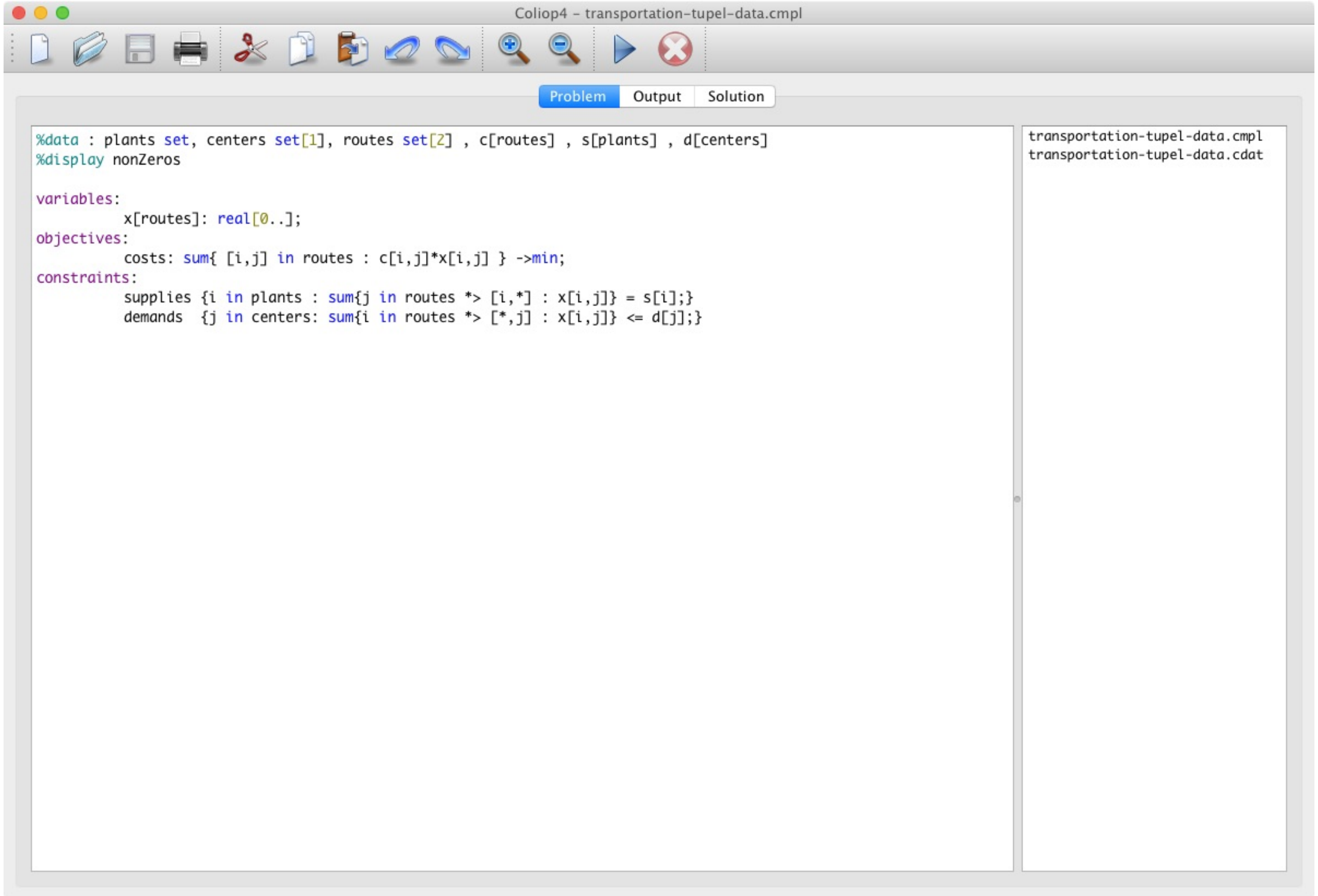
```
cmplServer -stop
```

5 User interfaces (Coliop and SolverStudio)

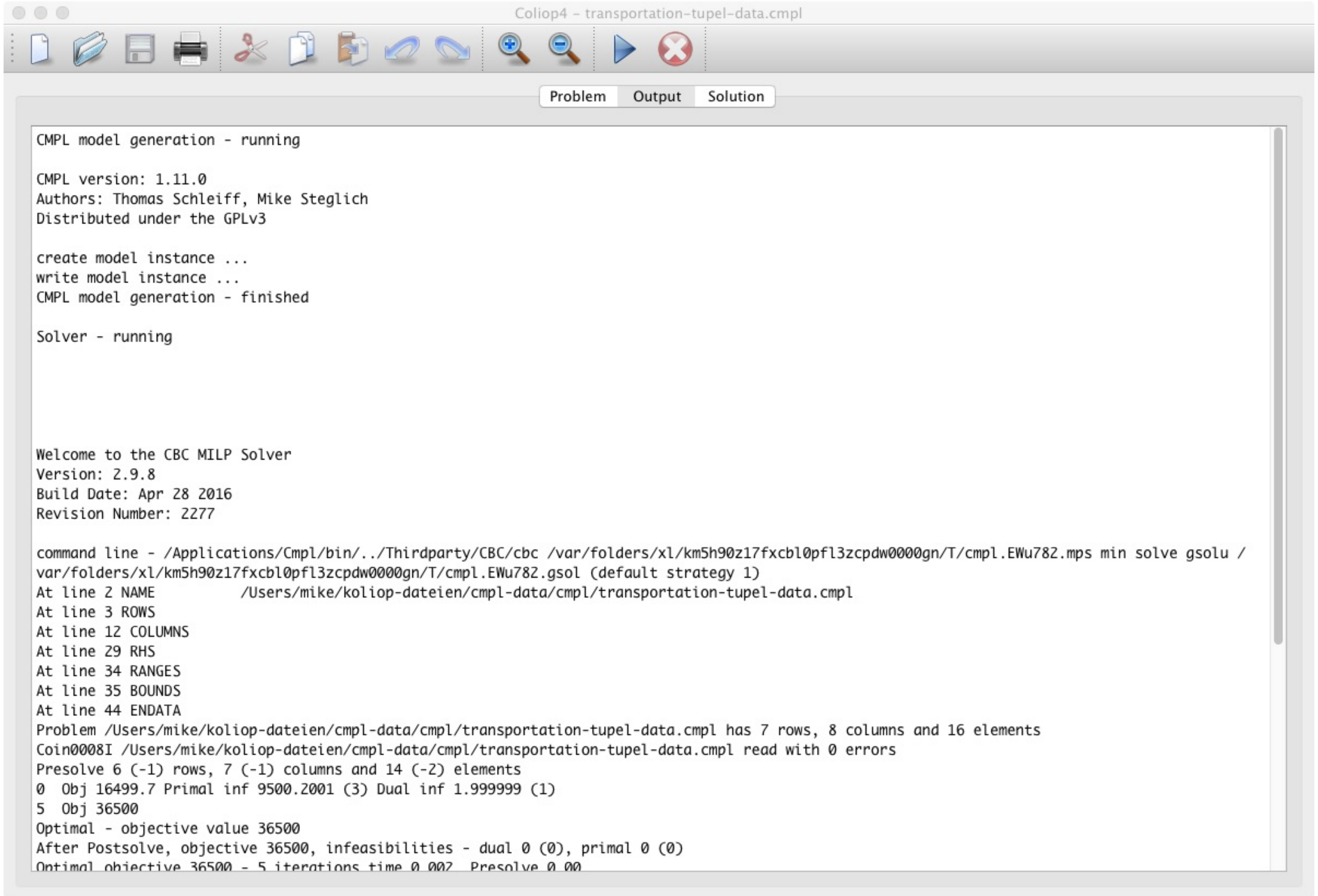
- **Coliop**
  - Coliop is an IDE (Integrated Development Environment) for CMPL intended to solve linear programming (LP) problems and mixed integer programming (MIP) problems. Coliop is an open source project licensed under GPL. It is written in C++ and is as an integral part of the CMPL distribution available for most of the relevant operating systems (OS X, Linux and Windows).

- **SolverStudio**
  - SolverStudio is an add-in for Excel on Windows that allows you to build and solve optimisation models in Excel using any of the following optimisation modelling languages:
    - PuLP, COOPR/Pyomo, AMPL, AMPL on NEOS server, GMPL (GNU MathProg Language), GAMS using the Gurobi Python modelling interface, SimPy, Julia and …
    - **CMPL and pyCMPL.**
  - SolverStudio allows you to create and edit your optimisation model without leaving Excel, and to save your model inside your workbook. You can also easily link data on your spreadsheet to sets, parameters, constants and variables used in the model. SolverStudio can run the model to solve the problem and then put the answer back onto the spreadsheet.

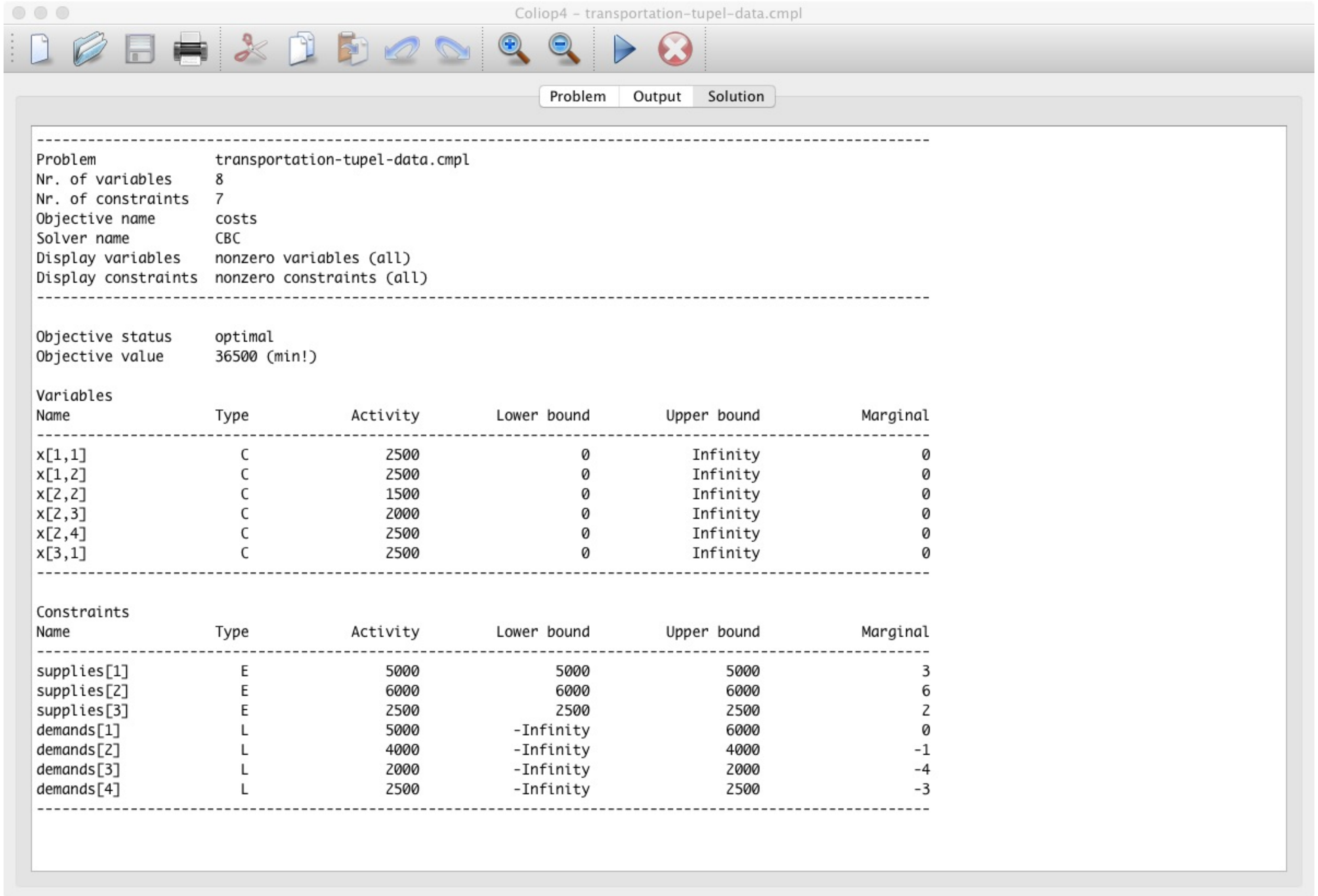
5 User interfaces (Coliop and SolverStudio)



5 User interfaces (Coliop and SolverStudio)

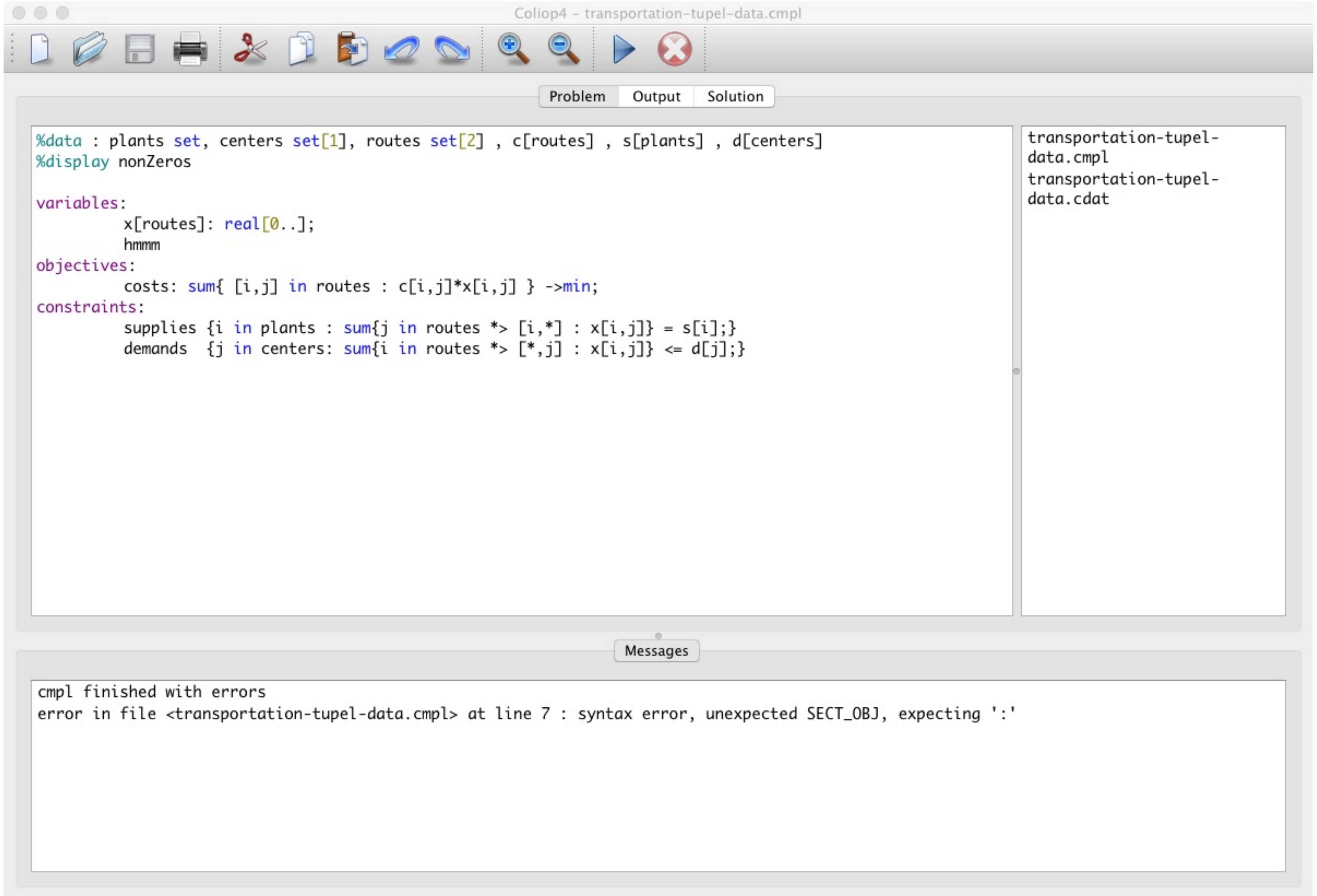


5 User interfaces (Coliop and SolverStudio)




```
---------------------------------------------------------------------------------------------------------
Problem              transportation-tupel-data.cmpl
Nr. of variables     8
Nr. of constraints   7
Objective name       costs
Solver name          CBC
Display variables    nonzero variables (all)
Display constraints  nonzero constraints (all)
---------------------------------------------------------------------------------------------------------

Objective status     optimal
Objective value      36500 (min!)

Variables
Name                 Type           Activity       Lower bound       Upper bound            Marginal
---------------------------------------------------------------------------------------------------------
x[1,1]                 C                2500                 0          Infinity                   0
x[1,2]                 C                2500                 0          Infinity                   0
x[2,2]                 C                1500                 0          Infinity                   0
x[2,3]                 C                2000                 0          Infinity                   0
x[2,4]                 C                2500                 0          Infinity                   0
x[3,1]                 C                2500                 0          Infinity                   0
---------------------------------------------------------------------------------------------------------

Constraints
Name                 Type           Activity       Lower bound       Upper bound            Marginal
---------------------------------------------------------------------------------------------------------
supplies[1]            E                5000              5000              5000                   3
supplies[2]            E                6000              6000              6000                   6
supplies[3]            E                2500              2500              2500                   2
demands[1]             L                5000         -Infinity              6000                   0
demands[2]             L                4000         -Infinity              4000                  -1
demands[3]             L                2000         -Infinity              2000                  -4
demands[4]             L                2500         -Infinity              2500                  -3
---------------------------------------------------------------------------------------------------------
```

5 User interfaces (Coliop and SolverStudio)



References

- Anderson, R. David, D.J. Sweeney, Th.A. Williams and M. Wisniewski (2014): An Introduction to Management Science - Quantitative Approaches to Decision Making, 2nded., Cengage Learning EMEA, Andover.
- Coulouris, G.F.; J. Dollimore, T. Kindberg, G. Blai (2012): Distributed Systems : Concepts and Design, 5th ed., Addison-Wesley.
- Fourer, R, J. Ma, R. K. Martin (2010): Optimization Services: A Framework for Distributed Optimization. Operations Research 58(6), 1624-1636.
- Hillier, F.S. und G.J. Lieberman (2010): Introduction to Operations Research, 9th ed., McGraw-Hill, New York et al.
- Kshemkalyani, A.D., M. Singhal, M. (2008): Distributed Computing – Principles, Algorithms, and Systems, Kindle ed., Cambridge University Press.
- Rogge, R. and M. Steglich (2007): Betriebswirtschaftliche Entscheidungsmodelle zur Verfahrenswahl sowie Auflagen- und Lagerpolitiken, in: Diskussionsbeiträge zu Wirtschaftsinformatik und Operations Research 10, Martin-Luther-Universität Halle-Wittenberg.
- St. Laurent, S., J. Johnston, E. Dumbill (2001): Programming Web Services with XML-RPC, 1st ed., O'Reilly.
- Vanderbei, R.J. (2014): Linear Programming: Foundations and Extensions, 4th ed., Springer, New York et al.
- Williams, H.P. (2013): Model Building in Mathematical Programming, 5th ed., Wiley, Chichester.

Thank you for your attention.

If you have any questions, please feel free to ask.