

# IE 477 - Introduction to ILOG CPLEX Optimization Studio

Serkan Turhan

December 5, 2019

This documentation is for the usage of ILOG CPLEX Optimization Studio (cited as *OPL* for short throughout this document: Optimization Programming Language, note that there are other IBM products that utilizes *OPL* in different programming environments such as Python, Java, C++, etc.) It describes the steps to install it, necessary packages to download in order to be able to run it and a general overview of its programming language. You can access the files to generate this pdf from this link.

Suggestion: while CPLEX is a good solver to analyze a problem and for trying different branch-and-bound techniques, I didn't have the chance to work with Gurobi back then, while preparing this documentation. If you are working with Python in your project, you will have easier time setting it up (specifically in the data handling and warm-start parts), it can also solve non-linear models, unlike CPLEX. More importantly, it is heavily documented and has active support forums, you can register an academic license with your bilkent email.

-Serkan

# Contents

|          |                                                             |           |
|----------|-------------------------------------------------------------|-----------|
| <b>1</b> | <b>Why CPLEX and OPL?</b>                                   | <b>3</b>  |
| <b>2</b> | <b>Installing the ILOG CPLEX Optimization Studio</b>        | <b>4</b>  |
| 2.1      | Windows issues while installing . . . . .                   | 9         |
| <b>3</b> | <b>Using ILOG CPLEX Optimization Studio</b>                 | <b>10</b> |
| 3.1      | Creating a project . . . . .                                | 10        |
| 3.2      | Turkish version issue . . . . .                             | 11        |
| 3.3      | File types . . . . .                                        | 12        |
| 3.4      | .dat files . . . . .                                        | 13        |
| 3.4.1    | Writing parameters . . . . .                                | 13        |
| 3.5      | .mod files . . . . .                                        | 13        |
| 3.5.1    | Defining parameters in the model . . . . .                  | 13        |
| 3.5.2    | Defining decision variables . . . . .                       | 14        |
| 3.5.2.1  | Defining a decision variable expression . . . . .           | 15        |
| 3.5.3    | Defining the objective function . . . . .                   | 15        |
| 3.5.4    | Defining the constraints . . . . .                          | 15        |
| 3.5.4.1  | Equalities and inequalities . . . . .                       | 16        |
| 3.5.4.2  | Summation . . . . .                                         | 16        |
| 3.5.4.3  | Constraint over a set . . . . .                             | 16        |
| 3.5.4.4  | “constraint 1 or constraint 2” type of constraint . . . . . | 16        |
| 3.5.5    | Readable format . . . . .                                   | 19        |
| 3.6      | .ops files . . . . .                                        | 21        |
| 3.7      | Configuration files . . . . .                               | 23        |
| <b>4</b> | <b>Checking the type of your problem</b>                    | <b>23</b> |
| <b>5</b> | <b>Final words</b>                                          | <b>23</b> |

# 1 Why CPLEX and OPL?

CPLEX has many benefits, one of those is that it is able to solve mathematical models really fast (depending on the type of problem) and easier to deploy/use compared to its alternatives: *OpenSolver*, *GLPK*, *Xpress*, etc. The main reason behind it is the heuristics and techniques that it utilizes before solving the model (pre-process) and while solving the problem (branch and bound operations). But these capabilities comes with a great price tag, so your company probably does not have the license for it, which means that you can use it only for the performance analysis of your heuristics, not for developing a solution for the company. *GAMS* uses CPLEX too, but licensing it requires more work than the *OPL*. Utilizing CPLEX while solving your mathematical models can save you great time, especially for NP-hard problems.

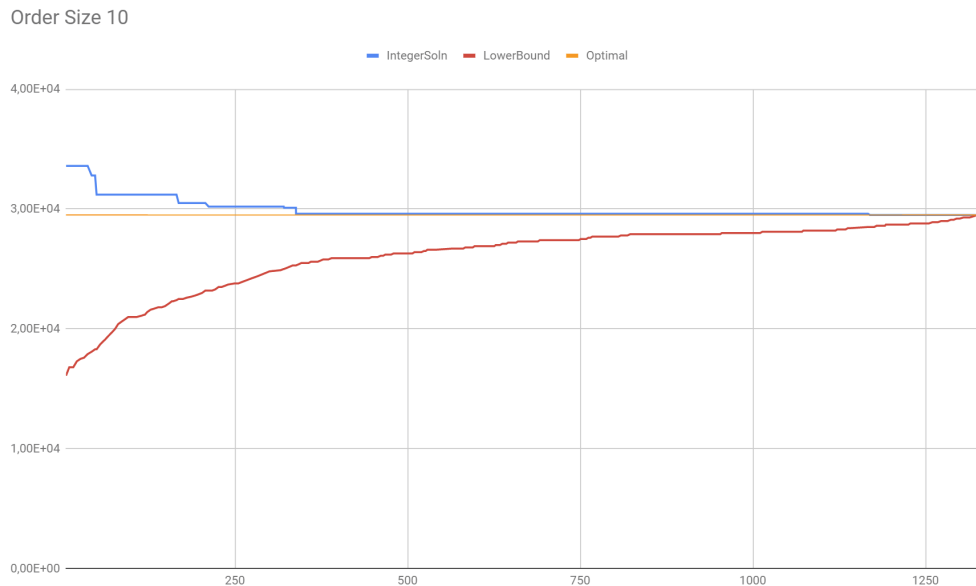


Figure 1: the objective value (minimize) vs. time (seconds) graph

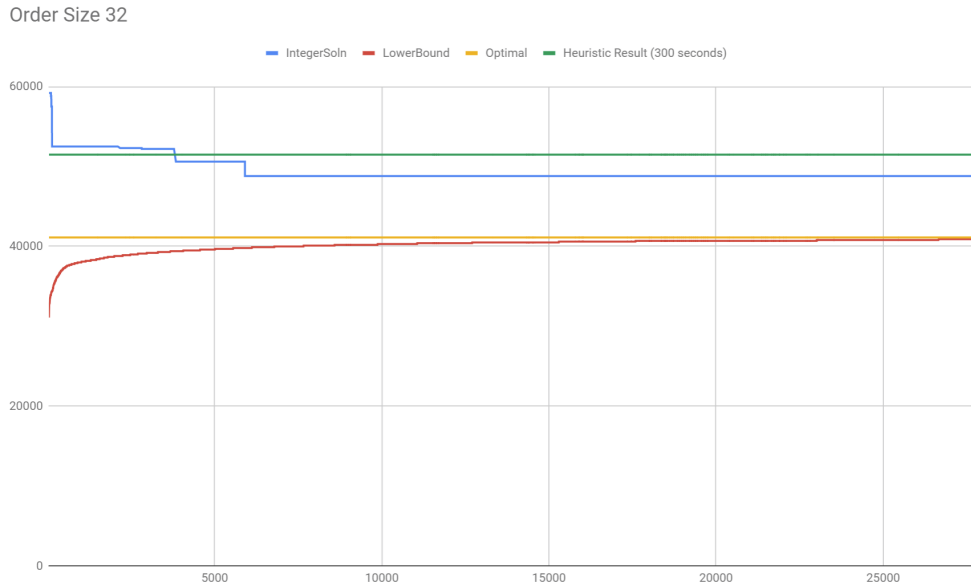


Figure 2: the objective value (minimize) vs. time (seconds) graph

In the figures 1 and 2, you can see the performance of *GLPK* on a vehicle routing problem, with two different problem sizes. Note that *CPLEX* solved these problems in less than 30 seconds! To give some depth on the problem, it was consisting around 20 general constraints and 7 decision variables (5 of which were binary) in three dimensions. If you have many data points, for which to run a mathematical model on, especially for integer programming problems, *CPLEX* can give you a result in a matter of seconds, while its alternatives give in a couple hours or may not give at all. By the way, you can utilize open-source solvers like this one in your project but as you can see, they are really slow at finding / improving the solutions than heuristical methods. So you need to apply some computational optimization and concurrency (the worst disadvantage of *GLPK* is utilizing only one core while solving, whereas *OPL* utilizes all of them) to be able to benefit from them.

## 2 Installing the ILOG CPLEX Optimization Studio

Having a Bilkent University associated e-mail is enough to register for an academic license of ILOG CPLEX. Note that if you download the installer on a computer and try to install it on a different computer, you will, most probably, get errors while running the program on that computer, at least this was always the case in our project. To download, go to this webpage.



Figure 3: Click "Get student and faculty editions for free"

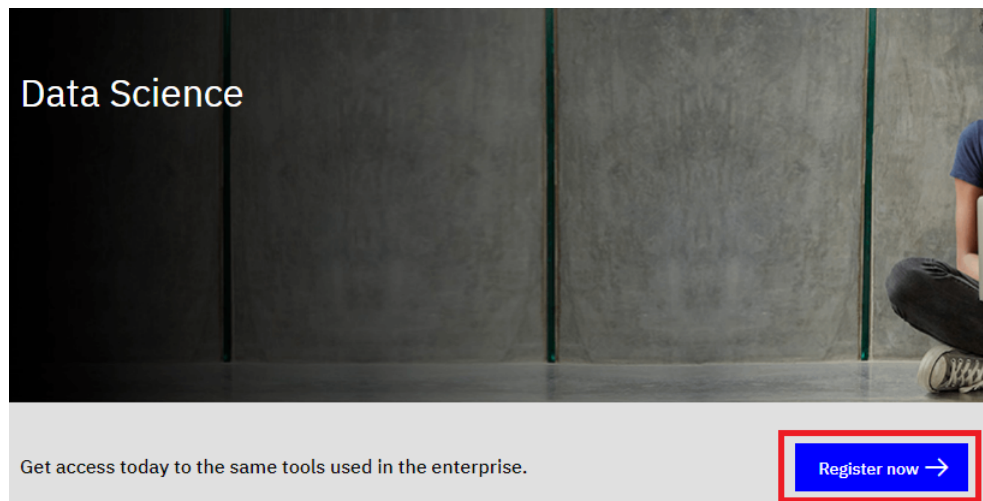


Figure 4: Click "Register Now", if it didn't direct you there after 3

## Enter your academic institution issued email to begin

Only the students and faculty of participating academic institutions are eligible to access this website. Please enter your academic institution issued email below to register.

Your academic institution issued email

name.surname@ug.bilkent.edu.tr

Submit

Figure 5: Enter your bilkent issued e-mail address

## Register Below

Complete the information below to register. In addition to the forms below you will need to register for an IBM ID to enroll in the program.

Academic Institution issued Email : name.surname@ug.bilkent.edu.tr

Academic Institution Name : Bilkent Üniversitesi - Bilkent University

Select one

- ☒ Student  
☐ Faculty

Current Degree

Select



Sought Degree

Select



Figure 6: Fill out the registration form

E-posta \*

ni\*\*\*\*\*@bilkent.edu.tr

Ad \*

Soyadı \*

Ülke veya bölge \* (?) ✓

Türkiye ✓

Parola ayarla \* Güçlü

.....

Figure 7: Finish the registration by entering your credentials and a password

## E-postanızı kontrol edin

Güvenliğiniz için kimliğinizi doğrulamanız gerekiyor.

ni\*\*\*\*\*@bilkent.edu.tr adresine 7 basamaklı bir kod gönderdik. Lütfen bu kodu aşağıya girin.

7 basamaklı kod girin ✓

1234567

Doğrula

Figure 8: Enter the code that's been sent to your registered e-mail

On the next page, you are automatically logged in to your created account, scroll down to see the products available to download for you. Note that if your project includes machine learning or some other stuff, you can utilize other tools of IBM as well.



### ILOG CPLEX Optimization Studio

Analytical decision support toolkit for rapid development and deployment of optimization models using mathematical and constraint programming. It combines an integrated development environment with the powerful Optimization Programming Language and high-performance CPLEX and CP Optimizer solvers.

[Download v12.8 →](#)

[Download v12.9 →](#)

Figure 9: Choose the version of the program to download

## Find by part number results

| eAssemblies (1)                   |                              |
|-----------------------------------|------------------------------|
| <a href="#">Expand all</a>        | <a href="#">Collapse All</a> |
| <a href="#">Download Director</a> |                              |
| <a href="#">HTTP</a>              |                              |

Figure 10: Select HTTP to avoid dealing with setting up the Download Director

☒ IBM ILOG CPLEX Optimization Studio 12.9 for Windows x86-64 Multilingual (CNZM1ML) - [View details](#)

Size

826MB

Date posted

8 Mar 2019

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

---

☐ IBM ILOG CPLEX Optimization Studio 12.9 for Linux x86-64 Multilingual (CNZM2ML) - [View details](#)

Size

624MB

Date posted

8 Mar 2019

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

---

☐ IBM ILOG CPLEX Optimization Studio 12.9 for Linux on System i/p Multilingual (CNZM3ML) - [View details](#)

Size

287MB

Date posted

8 Mar 2019

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

---

☐ IBM ILOG CPLEX Optimization Studio 12.9 for Linux on System z Multilingual (CNZM4ML) - [View details](#)

Size

350MB

Date posted

8 Mar 2019

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

Figure 11: Choose the installer according to your OS, OSX (for macs) is listed at the bottom

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

---

☐ IBM ILOG CPLEX Optimization Studio 12.9 for OSX Multilingual (CNZM6ML) - [View details](#)

Size

708MB

Date posted

8 Mar 2019

[License agreement](#)
[Download estimate](#)
[→ eAssembly](#)

---

By clicking the "I agree" button, you agree that (1) you have had the opportunity to read and understand the above license agreement(s) and multi-product package terms, if any, and (2) terms of the license agreement(s) govern this transaction. If you do not agree with the terms of the agreement(s), you will be unable to download the software.

☒ I agree
 ☐ I do not agree

Download now

Figure 12: Agree with the terms and click the Download Now button

After the download is finished, install it and you are done! (I could not provide screenshots for that since I deleted the installer long time ago and because I'm currently using mobile AP to connect to the internet, I don't want to exceed my quota by downloading it again 😞)

## 2.1 Windows issues while installing

It might result in an error or a warning before the installation in windows, prompting you to install a library before you can proceed. I can't quite remember because I already installed it, but I still have the installer of required library. Go to this webpage and download

it, if the webpage is not available, search for the **Microsoft .NET Framework 4.6.2** and install it from the official Microsoft website.

### 3 Using ILOG CPLEX Optimization Studio

The sections below describes how to create a simple 0-1 knapsack problem in *OPL*, you can follow the steps and replace the knapsack model and data with your mathematical model to solve it easily.

#### 3.1 Creating a project

Go ahead and launch the program, you can find the launcher by searching for `oplide` in the windows search bar (or the mac equivalent). In the launcher, follow the steps below:

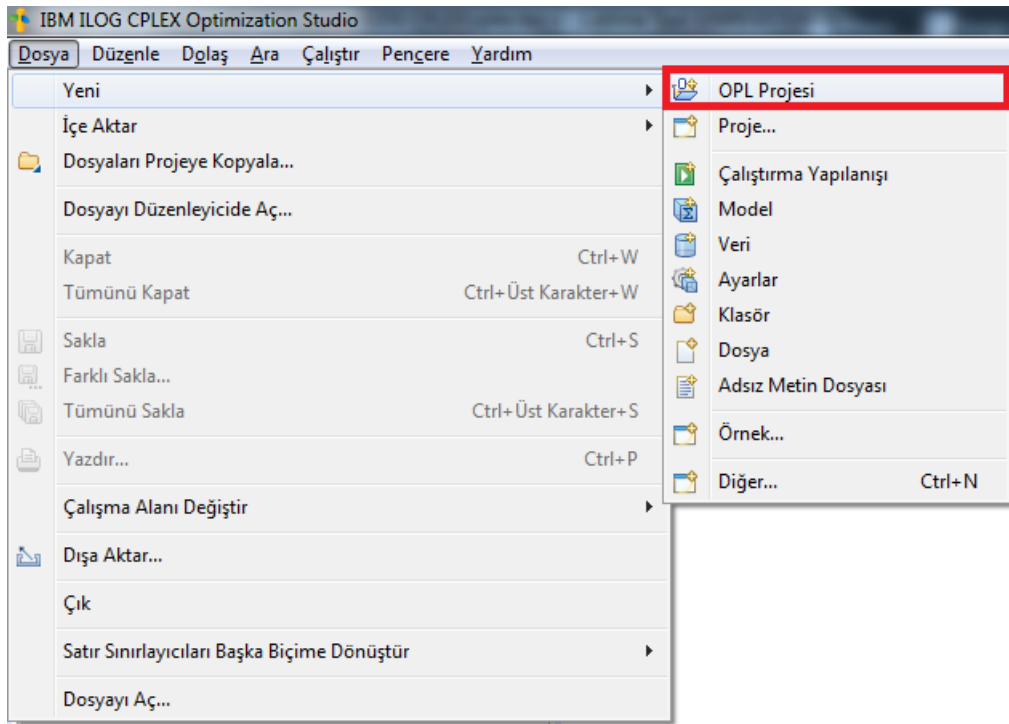


Figure 13: Navigate to create a new OPL Project

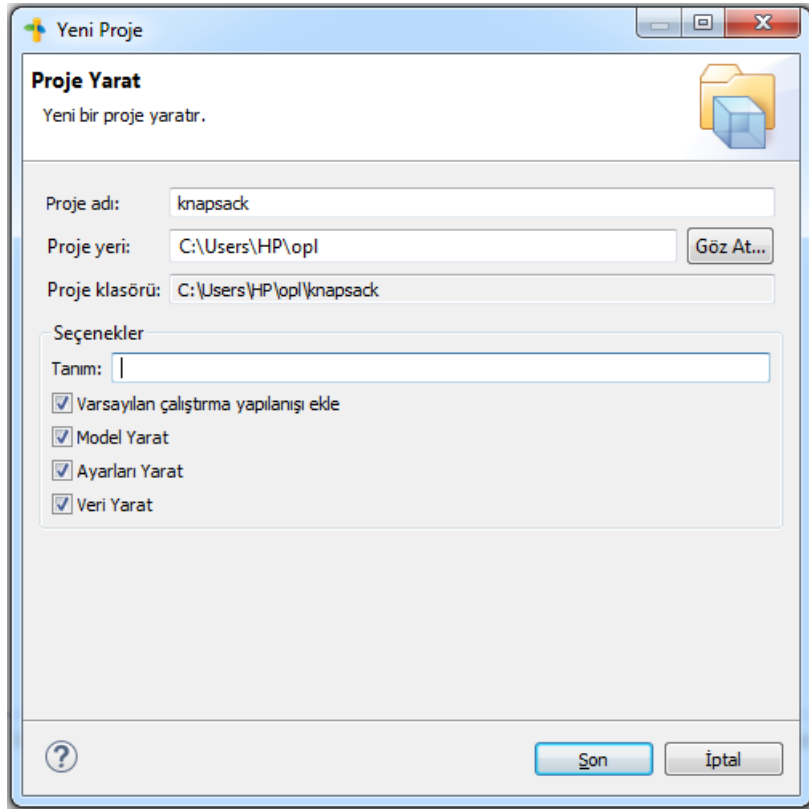


Figure 14: Name your project and make sure that all checkboxes are checked

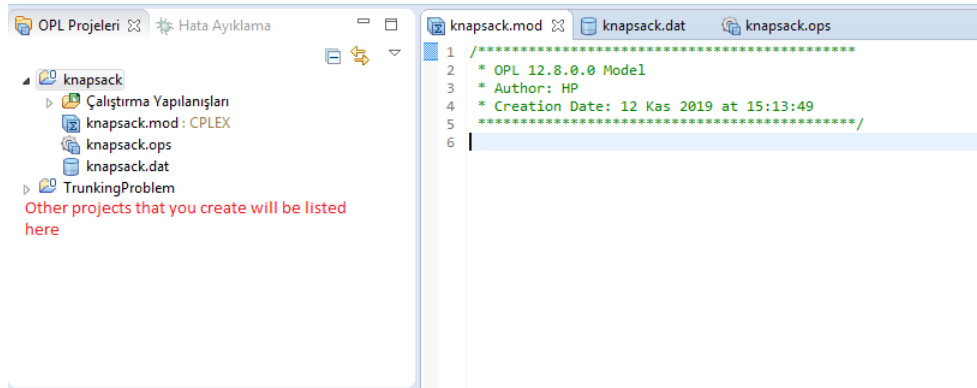


Figure 15: You should see something like this after creating it

As you can see, there four types of files.

### 3.2 Turkish version issue

This problem is specific to the Turkish version of the program. Since it creates the folders in Turkish, as seen from the figure 15, this creates an issue when you try to run the model,

because the program does not recognize the Turkish letters in the name under “Çalıştırma Yapılandırışları”. So what you need to do is as follows:

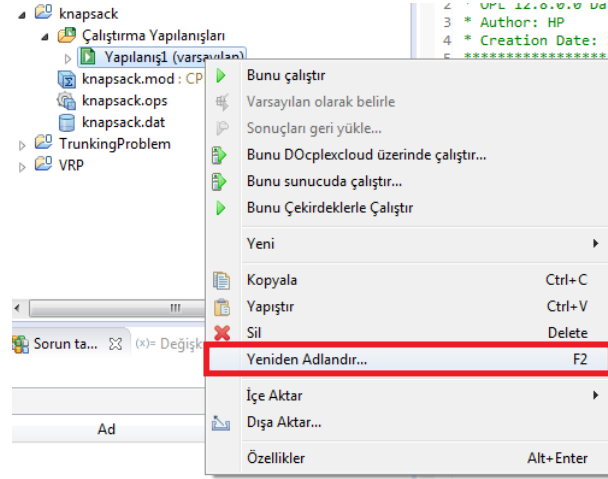


Figure 16: Choose the rename option

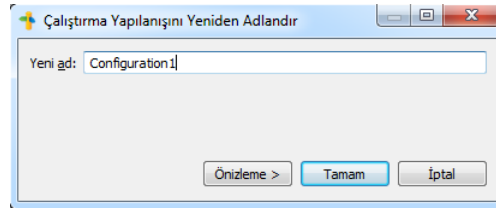


Figure 17: Choose a name that doesn't contain any non-english words

### 3.3 File types

In this section an overview of the files utilized in *OPL* are described. Note that this documentation does not cover in much detail, it is enough to write mathematical models but for extensive applications or for more detailed information, you should consult the official documentation provided by IBM, accessible [here](#).

- **.mod**: Model codes, constraints, decision variables
- **.dat**: Data files, includes parameters
- **.ops**: Customizing the solver engine, e.g. defining stop conditions
- **Configuration**: To configure run configurations, i.e. which combination of .mod / .dat / .ops files.

## 3.4 .dat files

These data files are crucial for your mathematical model to run smoothly, you can define the parameters explicitly in your model (which is described in section 3.5), but it is more organized if you use these data files.

### 3.4.1 Writing parameters

The examples for parameter declarations are in Code 1.

```
//scalar parameters
s = 5;
d = 10.42;
//vector parameters
vect = [3, 2, 5, 1, 3];
otherVect = [35.2, 13.63, 92.42];
//3x2 matrix
matrix = [[1, 3], [2, 3], [194, 23]];
//3x2x2 matrix
matrix3d = [[[22, 32], [3, 2]], [[...], ...], [[...], ...]] //not
               completely sure, never tried it
//or even strings
stringArr = ['`New York`', '`Ankara`'];
singleString = `IE477`;
```

Code 1: Defining various types of parameters

For our example, the Knapsack problem, go ahead and paste the Code 2 into your .dat file.

```
sackCapacity = 20;
itemCount = 10;
itemValues = [5.4, 3.2, 6.8, 1.0, 40.54, 5.6, 3.2, 8.9, 11, 9.5];
itemSizes = [5, 2, 1, 0.01, 13, 5, 6.7, 1, 2, 3];
```

Code 2: OPL code to create decision variables

## 3.5 .mod files

In this section, the steps to create a .mod file is explained.

### 3.5.1 Defining parameters in the model

The codes in these files are executed, so you write your models here. First, you should define your sets, the said sets here basically contain only the indices. The range object corresponds to a set in OPL, you can see the declaration in Code 3.

```
range items = 1..itemCount;
```

Code 3: OPL code to create a range

Then you set the parameters as in Code 4, or you can load them from the .dat files as in Code 5.

```
int sackCapacity = 20;
int itemCount = 10;
range items = 1..itemCount;
float itemValues[items] = [5.4, 3.2, 6.8, 1.0, 40.54, 5.6, 3.2, 8.9,
    11, 9.5];
float itemSizes[items] = [5, 2, 1, 0.01, 13, 5, 6.7, 1, 2, 3];
```

Code 4: OPL code to defining parameters without .dat file

```
int sackCapacity = ...;
int itemCount = ...;
range items = 1..itemCount;
float itemValues[items] = ...;
float itemSizes[items] = ...;
```

Code 5: OPL code to load parameters from .dat file

The main requirement here is that the variable names in .mod file and the .dat file should match. Putting three dots makes engine to search for the corresponding variable name in the .dat file and will result in an error if it cannot find it. Another point here is that, for the arrays, the sizes should match, otherwise it will result in an error as well. You should use a range variable inside the square brackets “[” “]”, and you can define multidimensional arrays as in Code 6.

```
int array2d[set1][set2] = ...;
float array3d[set1][set2][set3] = ...;
boolean array4d[set1][set2][set3][set4] = ...;
```

Code 6: Code to declare multidimensional arrays

### 3.5.2 Defining decision variables

You can define the decision variables as in the Code 7.

```
dvar float someNumericalDecVar;
dvar int someIntegerDecVar;
dvar boolean someBinaryDecVar;
dvar boolean some2dBinaryDecVar[set1][set2];
```

Code 7: OPL code to create decision variables

For the sake of our example, copy the Code 8 to your model file.

```
dvar boolean x[items];
dvar float sackValue;
```

Code 8: Code to create decision variables for Knapsack problem

**3.5.2.1 Defining a decision variable expression** You can utilize the decision variable expression for a more compact code. It basically means declaring a new decision variable and equating it to a some function of some decision variables, which you can later use this variable to reference this function. It can be used as in Code 9.

```
dexpr float fixedCost = reorderingCost + 2*travelCost;
```

Code 9: Code to create decision variables for Knapsack problem

Which is equivalent to the Code 10.

```
dvar float fixedCost;  
//some other model codes  
//...  
//in the constraints part of the code  
fixedCost == reorderingCost + 2*travelCost;
```

Code 10: Code to create decision variables for Knapsack problem

Using dexpr reduces the number of lines in your constraints section and makes it easier to make modifications on the code later on.

### 3.5.3 Defining the objective function

Defining an objective function is as easy as the other languages, for the examples, take a look at the Code 11. Partitioning your objective function by using dexpr will reduce the length of your objective function and make it more clear.

```
minimize fixedCost + reOrderCost; //minimization  
maximize profits - fixedCost - reOrderCost; //maximization
```

Code 11: Defining objective functions

The constraints will be after those. For our example, go ahead and paste Code 12 into your code.

```
maximize sackValue;  
subject to{  
  
}
```

Code 12: Defining objective function for our example

### 3.5.4 Defining the constraints

The constraints for the mathematical model will be written inside the `subject to` chunk. Let's start from the most basic operations for writing a constraint.

**3.5.4.1 Equalities and inequalities** Like the other languages, in *OPL*, for equalities and inequalities you do as in the Code 13. Just be careful about continuous (float) decision variables in strict inequalities, *OPL* doesn't like it when you use strict inequality for a continuous decision variable, so try to avoid it!

```
someEquation < someOtherEquation //strictly less than
someEquation <= someOtherEquation //less than or equal to
someEquation == someOtherEquation //equal to
someEquation >= someOtherEquation //greater than or equal to
someEquation < someOtherEquation //strictly greater than
someEquation != someOtherEquation //not equal to (note that this is
    also a strict inequality)
```

Code 13: Equality and inequality operators

**3.5.4.2 Summation** The examples of summation, including the conditional summation, is shown in Code 14.

```
sum(i in set1)x[i] >= someEquation; // summation
sum(i in set1, j in set2)x[i][j] >= someEquation; //2d summation
sum(i in set1)(sum(j in set2)x[i][j]) >= someEquation; //alternative 2d
    summation
sum(i in set1, j in set2 : i != j) x[i][j] >= someEquation; //
    conditional
```

Code 14: Summation operator examples

**3.5.4.3 Constraint over a set** You can define constraint(s) over set(s) as in Code 15

```
forall(i in set1) {
    x[i] >= someEquation;
    y[i] >= someEquation;
} // one set

forall(i in set1, j in set2){
    x[i][j] >= someEquation;
    y[i][j] != x[i][j];
} // two sets

forall(i in set1, j in set2 : i != j) {
    x[i][j] == y[i][j];
} // conditional constraint
```

Code 15: Defining constraints

**3.5.4.4 “constraint 1 or constraint 2” type of constraint** If you are facing a model where either constraint 1 or constraint 2 has to be satisfied, but not necessarily both at the same time. And you cannot do it by defining slack variables etc. You can do the following:

```
(x[i][j] >= someEquation) || (x[i][j] >= someOtherEquation);
```

Code 16: Either one of the constraints

Note that you should use this operator as a last resort, as your academic advisor(s), probably, won't approve this.

I believe these operators are sufficient for writing mathematical models in your project. If you need a more complex operation you should consult the documents using this link. Moreover, there are many example opl projects that you can make use of, although most of them are written in a little complicated way, you may find useful stuff in them. You can access the example projects by choosing the “examples” from the menu in figure 13.

Let's get back to our example project, now paste the constraints in Code 17 into the [subject to](#) chunk.

```
subject to {  
    sum(i in items)x[i]*itemSizes[i] <= sackCapacity;  
    sackValue == sum(i in items)x[i]*itemValues[i];  
}
```

Code 17: Defining objective functions

Finally our model is ready to run, follow these steps to run our example project:

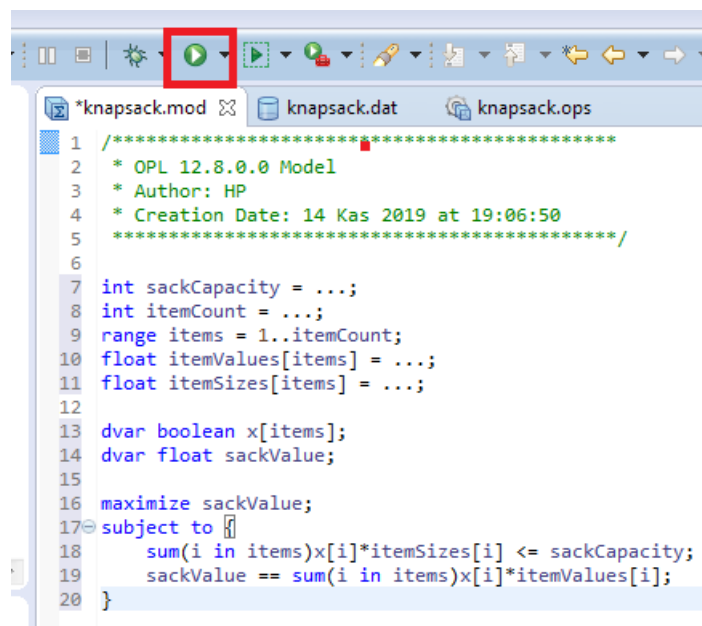


Figure 18: Click the run button located here

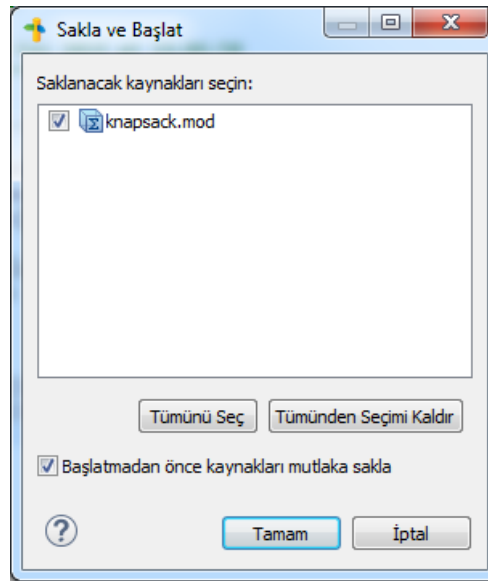


Figure 19: Accept and run, then you'll wait for program to initialize model and begin solving

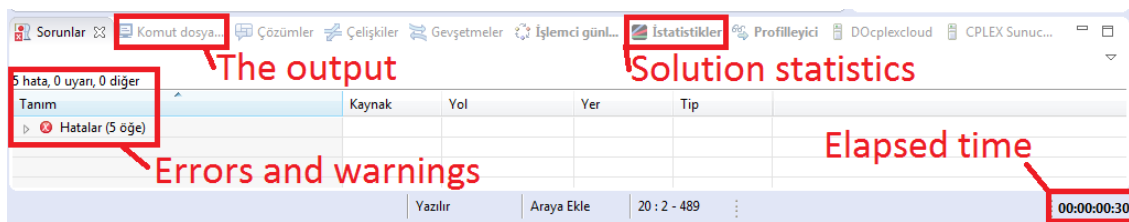


Figure 20: Check the box at the bottom of the page to see if there's any error. Warnings are okay though

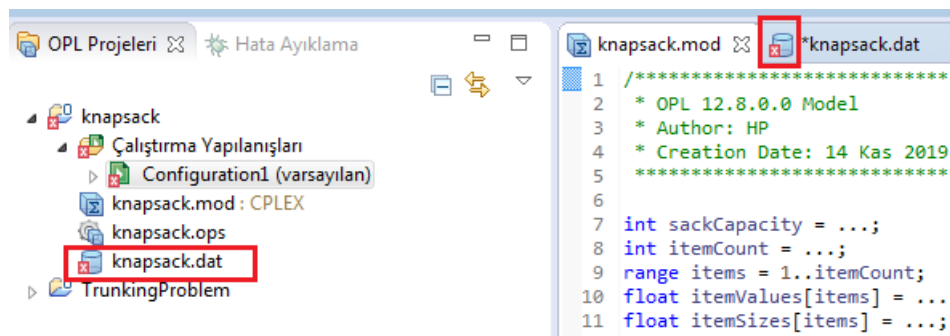


Figure 21: You'll notice which files produced the error. Configuration has an error sign as well because one of the files it contains produced an error

| Ad                            | Değer                                |
|-------------------------------|--------------------------------------|
| <b>Veri (5)</b>               |                                      |
| itemCount                     | 10                                   |
| items                         | 1..10                                |
| itemSizes                     | [5 2 1 0.01 13 5 6.7 1 2 3]          |
| itemValues                    | [5.4 3.2 6.8 1 40.54 5.6 3.2 8.9...] |
| sackCapacity                  | 20                                   |
| <b>Karar değişkenleri (2)</b> |                                      |
| sackValue                     | 76.74                                |
| x                             | [0 0 1 0 1 0 0 1 1 1]                |

Figure 22: After the solver is finished, you can notice the optimal objective and the optimal solution in this screen

### 3.5.5 Readable format

Unless you want to inspect each decision variable, after each time model is run, from the screen at figure 22, you should transform the solution into somewhat readable format. This can be accomplished by the usage of loops and the functions `writeln()` and `write()` inside the display chunk. For instance, for a TSP, if you have a two-dimensional decision variable, say  $x_{ij}$ , indicating whether the arc between nodes  $i$  and  $j$  is used, you can add / use the Code 18 to output the route in the following form:

Vehicle follows the route: 0 -> 2 -> 5 -> 3 -> 4 -> 1 -> 0

```
execute DISPLAY{
  write('\Vehicle follows the route: ')
  var currentNode = startingNode;
  for(i in nodeCount){
    if(x[currentNode][i] == 1) {
      write(i);
      currentNode = i;
      break;
    }
  }
  while(currentNode != startingNode) {
    for(i in nodeCount) {
      if(x[currentNode][i] == 1) {
        write('\ -> ', i);
        i = currentNode;
        break;
      }
    }
  }
  write('\ -> ', currentNode);
}
```

```
}
```

Code 18: OPL code to output route

For our example, go ahead and paste the Code 19 after the [subject to](#) chunk.

```
execute DISPLAY{  
  write('Items picked at the optimal solution:')  
  for(i in items) {  
    if(x[i] == 1) {  
      write(' ', i);  
    }  
  }  
  write(' ');  
}
```

Code 19: Code to display Knapsack problem solution

Now navigate to the section shown in figure 23, then re-run the model.

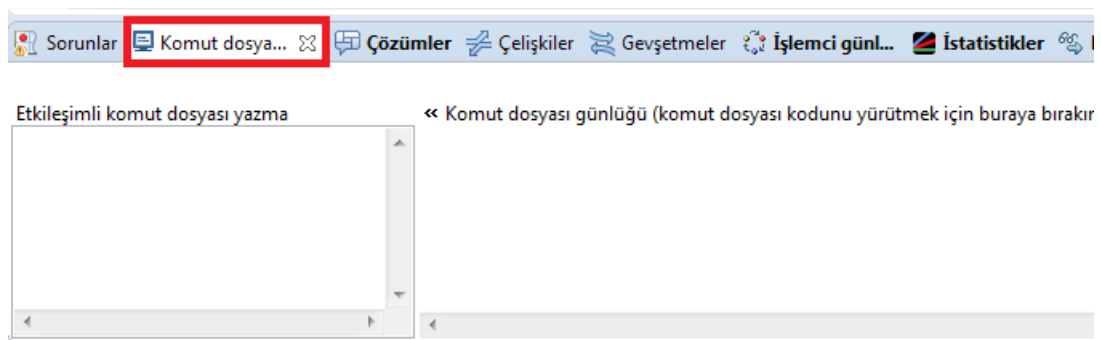


Figure 23: The outputs can be seen from this section

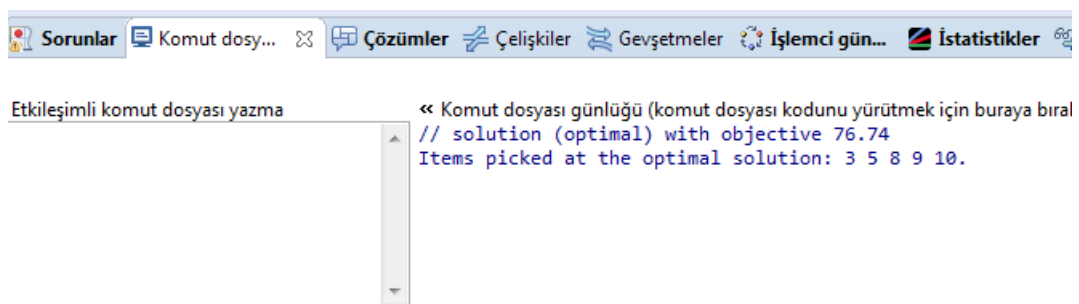


Figure 24: The output should look like this

The complete .mod file for the Knapsack example is in Code 20.

```
int sackCapacity = ...;  
int itemCount = ...;
```

```

range items = 1..itemCount;
float itemValues[items] = ...;
float itemSizes[items] = ...;

dvar boolean x[items];
dvar float sackValue;

maximize sackValue;
subject to {
    sum(i in items)x[i]*itemSizes[i] <= sackCapacity;
    sackValue == sum(i in items)x[i]*itemValues[i];
}

execute DISPLAY{
    write("Items picked at the optimal solution:");
    for(i in items) {
        if(x[i] == 1) {
            write(" ", i);
        }
    }
    write(".");
}

```

Code 20: Complete code for the Knapsack example

### 3.6 .ops files

In these file types, you can adjust many aspects of the solver engine. If your problem is NP-hard, the most important feature among those is the execution time limit, you can adjust it as in the figure 25.

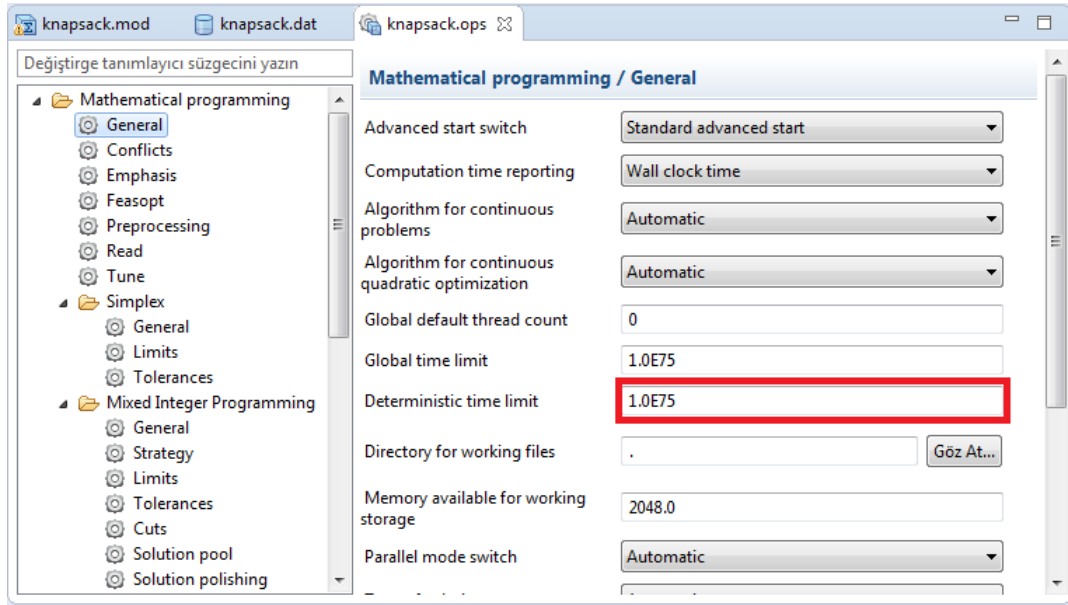


Figure 25: The time limit is set from this box

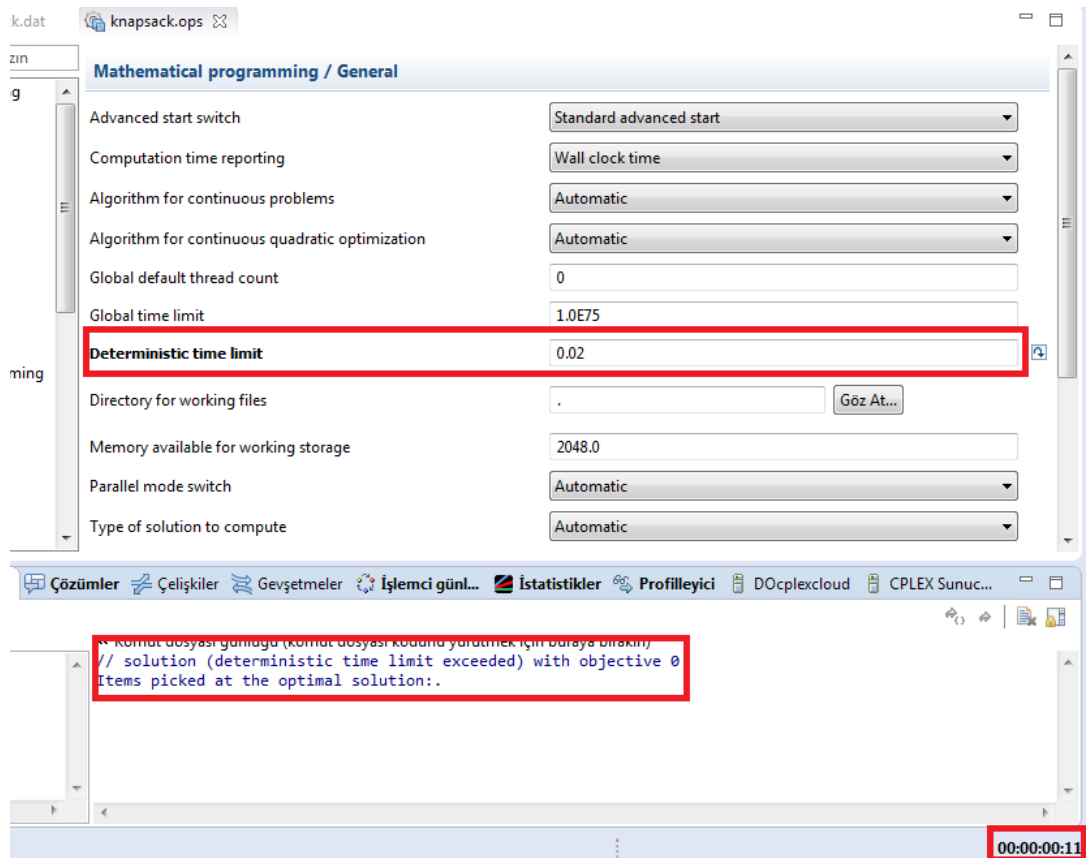


Figure 26: Run obtained with a time limit

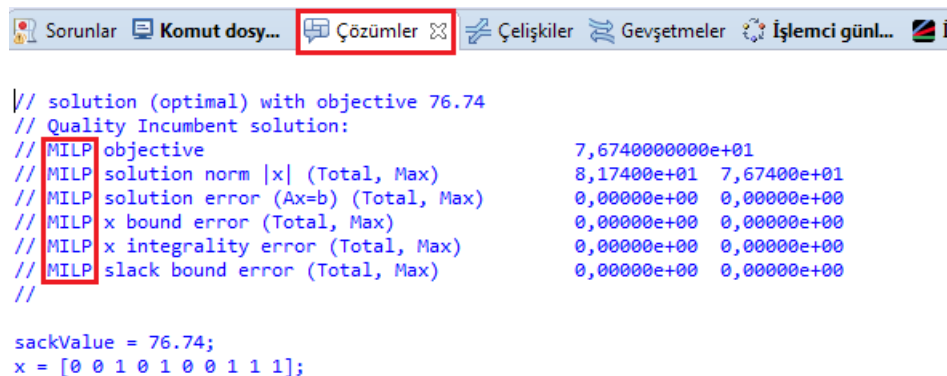
In my computer, the model would be solved within 20 milliseconds, and as you can see from the figure 26, setting a time limit of 2 milliseconds returns no solution. You can change other parameters of the solver engine but, unless you know what you are doing, it is best to leave them be.

### 3.7 Configuration files

These files hold the run configurations. They are useful if you have various different data with different, alternating models to run.

## 4 Checking the type of your problem

After your model is solved, from the “solutions” tab shown in figure 27, you can check which type of programming was used by the solver. Which indicates the type of your problem. The figure 27 shows the output for 0-1 knapsack model, Mixed Integer Linear Program was used. To check, you can add two binary (boolean) decision variables and simply multiply them at the objective, you’ll see that MIQP (Mixed Integer Quadratic Programming) is used to solve it. This is especially useful if you are not sure about the linearity of your model.



```
// solution (optimal) with objective 76.74
// Quality Incumbent solution:
// MILP objective 7,674000000e+01
// MILP solution norm |x| (Total, Max) 8,17400e+01 7,67400e+01
// MILP solution error (Ax=b) (Total, Max) 0,00000e+00 0,00000e+00
// MILP x bound error (Total, Max) 0,00000e+00 0,00000e+00
// MILP x integrality error (Total, Max) 0,00000e+00 0,00000e+00
// MILP slack bound error (Total, Max) 0,00000e+00 0,00000e+00
//
sackValue = 76.74;
x = [0 0 1 0 1 0 0 1 1 1];
```

Figure 27: Checking which program was used to solve model

## 5 Final words

For the most of the projects, the data will be provided in an excel file by the company. Transforming the information given into proper parameters for the *OPL* model might become an issue itself. For things to move smoothly, I would recommend that you utilize VBA to create your .dat file, then simply copy-pasting the output from VBA directly into the .dat file will save you great time, or even better, writing into the .dat file from

VBA. Of course this only applies if you have many parameter configurations to run the mathematical model on.

Good luck in your projects!