Introduction to OPL CPLEX

Torkel A. Haufmann

January 16, 2014

What is it?

- System for solving optimization problems
- OPL: Optimization Programming Language
- CPLEX: “Simplex in C”
- Various competing systems
    - Xpress-MP
    - GuRoBi
    - ...
- OPL CPLEX can be very useful in this course!

Anatomy of an optimization problem

Very informally, an optimization problem consists of two things:

1. A set of possible solutions to some problem.
2. A measure of "goodness" for any solution.

Anatomy of an optimization problem

Very informally, an optimization problem consists of two things:

1. A set of possible solutions to some problem.
2. A measure of "goodness" for any solution.

We are concerned with problems where both parts are described in *linear* terms. Hence, for us an optimization problem in n variables consists of:

1. A set in $\mathbb{R}^n$ defined by linear inequalities.
2. A linear function $\mathbb{R}^n \to \mathbb{R}$.

Describing an optimization problem

OPL is a *domain-specific language*, created for describing optimization problems.

What must we define?

1. Constants used in the problem.
2. Variables used in the problem.
3. The linear objective function.
4. The linear inequalities defining the feasible region.

Representing a problem

OPL separates the *model* and its *instance*.

- Model: `.mod` extension, describes the *structure* of a problem.
- Instance: `.dat` extension (or can be baked into `.mod`), describes the *data* in a problem.

Any linear program (in general form) has the same structure. Only the data changes!

In the OPL IDE, a model and data file are associated in a *run configuration*.

Defining constants and variables

OPL has two main kinds of data: *constants* and *decision variables*.

Constants:

- `float`
- `float+`
- `int`
- `int+`
- `string`

Defining constants and variables

OPL has two main kinds of data: *constants* and *decision variables*.

Decision variables:

- `dvar float`
- `dvar float+`
- `dvar int`
- `dvar int+`

Defining constants and variables

Often, we want to represent our data as arrays.

```
n = 4;
range vars = 1..n;
float+ b[vars] = [1, 2, 3, 4];
```

Defining constants and variables

Contrast:

```
dvar float+ x1;
dvar float+ x2;
dvar float+ x3;
dvar float+ x4;
```

```
range cols = 1..n;
dvar float+ x[cols];
```

Defining constants and variables

There is also a ... syntax for reading from a data file.

```
int n = ...;
int cols = 1..n;
dvar float+ x[cols];
```

We will get back to this later.

Defining the objective function

For example, let's maximize

$$6x_1 + 8x_2 + 5x_3 + 9x_4.$$

Defining the objective function

For example, let's maximize

$$6x_1 + 8x_2 + 5x_3 + 9x_4.$$

Without range:

```
dvar float+ x1;
dvar float+ x2;
dvar float+ x3;
dvar float+ x4;

maximize 6*x1 + 8*x2 + 5*x3 + 9*x4;
```

Defining the objective function

For example, let's maximize

$$6x_1 + 8x_2 + 5x_3 + 9x_4.$$

With range:

```
range cols = 1..n;
float c[cols] = [6, 8, 5, 9];
dvar float+ x[cols];

maximize sum(i in cols) c[i] * x[i];
```

Much more readable, and scales with `n`.

Defining the feasible region

Assume these constraints:

$$2x_1 + x_2 + x_3 + 3x_4 \leq 5,$$
$$x_1 + 3x_2 + x_3 + 2x_4 \leq 3.$$

Defining the feasible region

Assume these constraints:

$$2x_1 + x_2 + x_3 + 3x_4 \leq 5,$$
$$x_1 + 3x_2 + x_3 + 2x_4 \leq 3.$$

In OPL:

```
float A[rows][cols] = [[2, 1, 1, 3],
                       [1, 3, 1, 2]];
float b[rows] = [5,3];
dvar float+ x[cols];

(...)

subject to {
  forall (j in rows) {
    sum(i in cols) ( A[j][i] * x[i] ) <= b[j];
  }
}
```

Summarizing

A problem instance properly modeled in OPL consists of:

- A model file containing:
  1. Constant definitions (`float b = 3.0;`)
  2. Decision variable definitions (`dvar float+ x;`)
  3. An objective definition (`maximize ...`)
  4. Constraints (`subject to {... }`)
- A data file containing those constaints defined with `= ...;` in the model file.
- Optionally, other configuration options controlling the optimization.

Starting up the IDE

On a UiO Linux machine:

```
> oplide
```

From home:

```
> ssh -YC [username]@login.ifi.uio.no
> oplide
```

If you want to run the files `LP.mod` and `problem.dat` together without invoking the IDE you can use

```
> oplrun -v LP.mod problem.dat
```

If you do it this way you can write the files in whichever editor you prefer.