Capita AI, module 4.
Part A: Modeling an energy-aware scheduling problem

Tias Guns, *tias.guns@cs.kuleuven.be*

April 19, 2016

This module of the Capita AI course consists of two parts: the first, small, part consists of modeling a scheduling problem where the goal is to assign jobs to servers such as to minimize the total energy cost; the second part is the main part: to develop a framework that forecasts expected energy prices and solves the energy-aware scheduling problem such that the *real* energy cost would have been minimal.

Deadlines

- **Tuesday 19/04/2016 at 20:00** the latest: a file called *energychallenge-YOURTEAMNAME.mzn* by email to `tias.guns@cs.kuleuven.be`
- **Friday 20/05/2016 at 20:00** the latest: a scientific report of at most 4 pages with sections: 1) abstract summary of your findings (1-2 paragraphs), 2) method (overview of scheduling approach, learning approach and their integration), 3) results and discussion (of interesting experiments you did), 4) conclusions and future work (if you had more time, you would investigate...); as well as a zipfile of all your code, data and scripts.

The assignment should be made in the same teams as the previous assignments.

1 ICON energy challenge

The problem is based on the ICON Energy Challenge:
`http://iconchallenge.insight-centre.org/challenge-energy`

Here is the full description of the original challenge:

> ”You are running a cloud computing service, where customers contract to run computing services (tasks). Each task has a duration, an earliest start and latest end, and resource requirements for CPU, Memory and I/O attributes. The tasks can be scheduled on one of multiple servers, each server has a limited

capacity for the CPU, memory and I/O attributes. Multiple tasks can run concurrently on the same machine if the total resource consumption for all attributes is below the respective capacity. All tasks must be scheduled within their release and due dates, these dates are set so that no task stretches over midnight between two days. Tasks can not be interrupted, once started, they must run for their given duration. If a machine is used by a task, it must be running at that time. In addition to the cost of running the allocated tasks, the machine consumes some idle power if it is on. Every time a machine is switched on or off, a start-up resp. shut-down cost must be paid. All machines are off at the beginning of the planning period, all machines must be off at the end of the planning period.

The price of electricity for the data centre is a real-time price, and varies throughout the day. The actual price is not known in advance, a forecast must be used to generate a schedule. The total cost of the schedule is determined after the fact by applying the actual price of electricity to the energy consumption in each time period. One forecast of the price is given by the organizers. However there may be a large discrepancy between the forecast and actual price, illustrated in the figure, offering the opportunity to generate better forecasts based on historical data for demand and prices, and previous forecast information. Note that a forecast with a low error is not automatically guaranteed to lead to a schedule with a low overall cost.

You use the provided price forecast to generate your best possible schedule. All tasks must be scheduled, the cost of the schedule is given by the actual price for electricity. The validity and cost of the schedule generated is tested by a solution checker. You generate your own price forecast, and use a provided scheduling system to generate a schedule. The cost of the solution will depend on the quality of your price forecast. You generate both the price forecast and a scheduling system. You provide a schedule, its cost will be determined by a solution checker."

In the challenge, the scheduling problem to solve was really hard. This forced all participants to focus only on the scheduling part, instead of the more novel interactions between the learning and the scheduling.

For this module of Capita AI, we will redo the ICON Energy Challenge, but we will use simplified scheduling instances so that the scheduling part is a lot easier and focus can be put on the interaction.

Part A is about writing a model of the scheduling problem in the MiniZinc language. **Part B** will be about developing a learning method and experimenting with different approaches.

I provide (simplified) scheduling instances of different hardness, and the historic data of the ICON challenge. I also provide some basic python scripts that can help you start faster.

2 Part A: minizinc model of the scheduling problem

1. Problem description Please read the description PDF of the original challenge at: `http://iconchallenge.insight-centre.org/task.html`

For this course, we will use the following simplifications:

- We ignore start-up, idle, and shut-down costs. These aspects do NOT have to be modelled. We assume the machines are always on, and do not incorporate these cost.
- We will not use the scheduling instances of the ICON challenge. Instead, I provide you with simpler instances (lower time resolution, one resource only (CPU) and not too big).

2. MiniZinc and MiniZinc IDE The problem should be modelled in MiniZinc, this is a general-purpose modeling language for combinatorial problems. It has its own pre-configured IDE and decent documentation. You will have to learn the syntax, and how to model problems conceptually in such a language.

For this, follow the getting-started part of `http://minizinc.org`, that is, download the MiniZinc IDE (version 2.0.13 or later) and read (and run!) the MiniZinc Tutorial. I highly recommend reading up to page 25 in detail, and to pick what you find interesting or think you need afterwards. *(one user has reported problems when the MiniZinc IDE is in a path with spaces on a mac)*

3. Modeling and solving the test instance Once you are familiar with the MiniZinc IDE and the models described in the MiniZinc tutorial, you can start on creating your own model for the energy challenge. To get started, download the capita4_A.zip pack from toledo. The file 'energy-skeleton.mzn' is a skeleton minizinc file which already contains the data definitions, the variables and the print statement, but no constraints yet. Writing the constraints, based on the description of the challenge but without the start/idle/shutdown costs, is what you have still have to do.

The file energy-test.dzn is a MiniZinc datafile that you can use to test your model. It is a trivial instance with one machine and one task and a forecast. Note that because we are dealing with floating point numbers, you can only use Gecode and G12 MIP as solvers. You can configure both the datafile and the solver in the 'Configuration' tab of the MiniZincIDE.

When running the skeleton model with the test datafile, you will get the following output:

```
Cost=0.0
----------
==========
```

After you modelled the constraints correctly, you should get the following output:

```
Machine=1,Start=9,Task=1
Cost=267.053664
----------
==========
```

4. Testing on a wide range of instances If the above works, you can now test your model (for correctness and efficiency) on a range of instances. I have provided the `mzn-runcheck.py` script for this, but first you should configure your computer such that the root directory of the MiniZinc IDE (which contains the minizinc, mzn-gecode and mzn-g12mip binaries) is on your `PATH`, such that you can run the following commands from command-line:

```
mzn-gecode energy-teamtias.mzn energy-test.dzn
mzn-g12mip energy-teamtias.mzn energy-test.dzn
```

Alternatively, you can tell mzn-runcheck.py where the MiniZincIDE dir is, by hardcoding it in the file or adding the argument `--mzn-dir <MiniZincIDE dir>` below.

Once this is done, you can test your model on non-trivial instances in the ICON challenge format. For now, we will always use the `forecast.txt` file in the main directory. You can now use the `mzn-runcheck.py` script as follows:

```
./mzn-runcheck.py energy-teamtias.mzn load1/day01.txt forecast.txt
```

If you add `-p` to enable pretty printing, the output will be:

```
== Machine 0 ===================================
--------XXXXXXXXXXXXXXXXXXXXXXX                - :: Task 1
----     XXXXXXXXXXXXX-------------------------- :: Task 8
== Machine 1 ===================================
--------------------XXXXXXXXXXXXXXXXXXXXXXXXX  - :: Task 2
-------------------XXXXXXXXXXXXXXXXXXX         - :: Task 4
-  XXXXXXXXXXX---------------------------------- :: Task 6
-----XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX              :: Task 7
--       XXXXXXXXXX                  ----------- :: Task 9
== Machine 2 ===================================
-        XXXXXXXXXX      ----------------------- :: Task 0
----    XXXXXXX                           ------ :: Task 3
-------------------XXXXXXXXXXXXXXXXXXX           :: Task 5
f_instance; f_forecast; cost_forecast; cost_actual; time
load1/day01.txt; forecast.txt; 10264.3681; -; 0.65
```

The pretty-printing visualisation has one symbol for each timeslot, it is a '-' if the timeslot is before or after the earliest/latest start time of the task and an X if the task is running at that time.

The script has a number of other options, to help with debugging your model or running of experiments. Run `./mzn-runcheck.py --help` for an overview.

There are 8 load packs, and your model should give a solution for all these models. If not, there is a bug in your model. load1 is the easiest pack, load8 will take most computation time.

By the deadline mentioned at the top of this page, you should send me your working MiniZinc file. (in the next part, you will combine your own forecasting method with your scheduling modle, and you will be well equipped to change the model if need be).