Modelling for Combinatorial Optimisation (1DL451) and Part 1 of Constraint Programming (1DL442) Uppsala University – Autumn 2024 Cheatsheet: MiniZinc Backends & Experiment Script

Gustav Björdal, Pierre Flener, and Frej Knutar Lewander

30th August 2024

1 MiniZinc Installation and Command-Line Interface

On the Windows computers in the basement of Ångström house 10, start the ThinLinc client from the start menu, set "Server" to `thinlinc.student.it.uu.se`, set "Username" to your user identity, set "Password" to your password A, and press "Connect".

Our script `first_install.sh` adds three aliases in your `.bashrc` file in your home directory:

```
> cd /it/kurs/consprog/minizinc/
> ./first_install.sh
```

Note that in order to execute the aliases in terminal tabs you had open before you executed `first_install.sh`, you must for each terminal tab execute the command:

```
> source ~/.bashrc
```

There should now be three new aliases in your `.bashrc` file in your home directory that can be executed in your terminal: `minizinc`, `minizinc-ide`, and `run_backends`.[1]

Since non-interactive terminals do not automatically process the `.bashrc` file in your home directory, the aliases cannot be used in commands that in turn run other commands (such as the `timeout` command) or script files, but must instead be replaced by their associated command (such as `/it/kurs/consprog/minizinc/run_backends.sh` for the `run_backends` alias). In order to use the Gurobi backend in the MiniZinc integrated development environment (Section 2) and the MiniZinc command-line interface in a non-interactive terminal, you must *first* use the following command:

```
> module load gurobi
```

Our experiment script (Section 3) runs this command *automatically*.

Our MiniZinc command-line interface (CLI), with backends of *all* solving technologies that are to be used in this course (note that the official CLI just has a subset thereof), is at

```
/it/kurs/consprog/minizinc/MiniZincIDE/bin/minizinc
```

and, after running our `first_install.sh` script, can be accessed with the `minizinc` alias. See Table 1 on the second-last page for information about our installed backends. Run MiniZinc from the command line using:

[1] The Python source code for the `run_backends` script is accessible at GitHub.

```
> minizinc
```

Get help that is specific to a backend by using its name in Table 1:

```
> minizinc --help <backend>
```

For more information on the official MiniZinc CLI, see its documentation.[2]

2 MiniZinc Integrated Development Environment

Our MiniZinc integrated development environment (IDE),[3] with backends of *all* solving technologies that are to be used in this course (the official IDE just has a subset thereof), is on the ThinLinc computers of the IT department at `/it/kurs/consprog/minizinc/MiniZincIDE/` and, after running our `first_install.sh` script, can be accessed with the `minizinc-ide` alias. Feel free to use our IDE (if not the official IDE) while designing your models, but test each model under *all* solving technologies, even when still designing it. An IDE is *not* suitable for running the experiments we ask for in the assignment and project reports, as for this you ought to run the backends from our CLI or, much more conveniently, using our script of Section 3.

3 Running a Batch of Experiments

Our `run_backends.sh` script (accessed with the `run_backends` alias after running our `first_install.sh` script) *both* runs a batch of experiments on all the backends we installed on the ThinLinc computers of the IT department *and* generates a results table in LaTeX format. It can run either a set of `.dzn` files or by increasing values of some parameter, as discussed in the following subsections. Type `run_backends --help` to see its flags.

3.1 Setting Everything Up

To run our experiment script, move to the directory where your `.mzn` model file and a directory with your `.dzn` datafiles are located. Assume you have a directory called `myModels` in your home directory, with a model file called `tilePacking.mzn` for a tile packing problem and a directory called `dataDir` with your datafiles. Navigate to this directory:

```
> cd ~/myModels/
> ls
dataDir/
tilePacking.mzn
> ls dataDir
instance_4.dzn
instance_5.dzn
instance_6.dzn
```

3.2 Extracting Values

Assume you want to report the width, height, and area of the best-found bounding box in a table, where there are variables `w` and `h` in the model corresponding to the width and height of the bounding box, and the area is to be minimised. For our experiment script to extract these

[2] https://docs.minizinc.dev/en/stable/command_line.html
[3] https://docs.minizinc.dev/en/stable/minizinc_ide.html

values, use the `--vars` flag with the names of these variables; *for optimisation problems, the objective value is extracted automatically, so you should* not *specify it*:

```
... --vars w h ...
```

The values of some variables cannot be extracted unless explicitly annotating they are output variables in the MiniZinc model. For example, we declare that the variable `x` is an output variable with the `add_to_output` annotation in order for our script to be able to extract it:

```
% assuming n, y, and z are declared elsewhere:
var 1..n: x :: add_to_output = y+z;
```

Each of the space-separated words must be the exact name of a variable in the MiniZinc model.

3.3 Running Experiments over a Parameter Range

Assume you want to run the model for an increasing number `n` of tiles. In order to allow our script to control this, declare a parameter, say `n`, in your model:

```
int: n;
```

The following command runs the `tilePacking.mzn` model (assumed to be in the current directory) for values of `n` between 3 and 15, by increments of 5, with a time-out of 60 seconds, and writes the results in LaTeX format into a file called `results.tex`:

```
> run_backends --vars w h -r n 3 15 5 -t 60s \
  -o results.tex tilePacking.mzn
```

The \ character means a line break since the command does not fit on one line in this document: you should *not* include it but write everything on one line. After importing the output file into a LaTeX `tabular` environment using the LaTeX command `\input{results.tex}`, you get something like Table 2, except for the left column. The meanings of the flags can be seen when typing `run_backends` at the command line. Note that `-r <param> <start> <stop> <inc>` increments `<param>` by `<inc>` until it is greater than `<stop>`, but it does not necessarily run the instance where `<param> = <stop>`. In the example `"n 3 15 5"` above, the script only runs for the instances `n=3`, `n=8`, and `n=13`.

Each instance is currently only run *once* on each backend, so no statistics are performed on the results of multiple runs, even though this ought to be done, especially for backends that perform randomisation, such as `yuck`.

3.4 Running Experiments on a Set of Datafiles

Our experiment script can also run the model for a set of datafiles (as required by Assignments 2 and 3). Assuming you have three datafiles called `instance_4.dzn`, `instance_5.dzn`, and `instance_6.dzn` in a directory called `dataDir` in the same location as your `tilePacking.mzn` model, you can run the model for those datafiles, with a time-out of 60 seconds, and write the results in LaTeX format into a file called `results.tex` using the following command:

```
> run_backends --vars w h -d dataDir/*.dzn -t 60s \
  -o results.tex tilePacking.mzn
```

Recall that the \ character means a line break since the command does not fit on one line in this document: you should *not* include it but write everything on one line. After importing the output file into a LaTeX `tabular` environment using the LaTeX command `\input{results.tex}`,

you get something like Table 2. The meaning of the flag `-d` is shown upon typing `run_backends` at the command line.

Recall that each instance is currently only run *once* on each backend, so that no statistics are performed on the results of multiple runs, even though this ought to be done, especially for backends that perform randomisation, such as `yuck`.

3.5 Running Experiments Overnight

If you want to run your experiments while being logged out of a chosen ThinLinc computer[4] of the IT department, then log into one of the ThinLinc computers using `ssh` and use the `screen` command to start a detached session. Open a terminal and connect via `ssh` to one of the ThinLinc computers:

```
> ssh username@<hostname>.it.uu.se
```

Once logged in, write:

```
> screen -Rd
```

This changes your terminal to what looks like a fresh terminal. From here you can perform the steps described in the previous subsections in order to start your experiments.

Note: If you are in a computer lab and start a detached `screen` session directly in a terminal, without first doing an `ssh`, then the session will *not* persist after you log out and your experiments will be aborted.

You should put a reasonable global time-out on the entire batch of your experiments so that they do not run for too long! You can set for instance a 1-day time-out using `timeout 1d` when you start our experiment script:

```
> cd ~/myModels
> timeout 1d /it/kurs/consprog/minizinc/run_backends.sh \
  --vars w h -d dataDir/*.dzn -t 60s -o results.tex \
  tilePacking.mzn
```

Once you have started our experiment script, you can exit the `screen` session by pressing `Ctrl-a Ctrl-d`. This will return you to the terminal from where you started `screen`. If at this point you log out, then the script will keep on running.

If you want to check the progress of the experiments and possibly terminate the experiments (say because you have in the meantime found some improvements to your model), then open a terminal and run `screen -Rd` again: this gets you back to where you left off and you can kill the experiment script using `Ctrl-c`.

Note: The `screen` session is *local* to the ThinLinc computer you had logged into, so if you started it from `siegbahn.it.uu.se` and then log into `fredholm.it.uu.se`, then you will *not* be able to reopen the `screen` session. However, when the script finishes you can access the results from any ThinLinc computer, provided you have output them to a file.

[4] https://www.it.uu.se/datordrift/maskinpark/linux

4 Error Reports

This is a working document that we will update during the course if bugs or features are discovered. *Always* first check your results against some runs from the CLI or IDE in order to see that things look about right. If our experiment script reports `ERR` for any run, then this most likely means that there is something wrong with your model. Do checks at the CLI or IDE in order to determine what is wrong. If you have strong reasons to suspect a bug on our side, then please report it to the COCP helpdesk as soon as possible.

Backend identifier	**Backend name**	**Proves Optimality**	**Miscellaneous and Known Bugs**
`gecode`	Gecode	ultimately	Good all-round backend that supports all MiniZinc features.
`cp-sat`	CP-SAT	ultimately	–
`chuffed`	Chuffed	ultimately	–
`gurobi`	Gurobi	ultimately	Generates at each run a log file that it is safe to remove. If you run it via our MiniZinc CLI, then you must first run the `module load gurobi` command.
`coin-bc`	COIN-BC	ultimately	Use the commercial Gurobi Optimizer if possible.
`highs`	HiGHS	ultimately	Use the commercial Gurobi Optimizer if possible.
`yuck`	Yuck	occasionally	–
`picatsat`	PicatSAT	ultimately	–

Table 1: MiniZinc backends accessible on the ThinLinc computers of the IT department through our CLI `/it/kurs/consprog/minizinc/MiniZincIDE/bin/minizinc` and our IDE `/it/kurs/consprog/minizinc/MiniZincIDE/MiniZincIDE.sh`.

Backend	Chuffed		CP-SAT		Gecode		Gurobi		PicatSAT		Yuck	
n	w,h,obj	time	w,h,obj	time	w,h,obj	time	w,h,obj	time	w,h,obj	time	w,h,obj	time
instance_4	7,5,**35**	**4**	5,7,**35**	640	7,5,**35**	623	7,5,**35**	765	7,5,**35**	712	5,7,**35**	t/o
instance_5	12,5,**60**	**6**	5,12,**60**	617	12,5,**60**	604	12,5,**60**	679	12,5,**60**	740	12,5,**60**	t/o
instance_6	11,9,**99**	**12**	9,11,**99**	624	11,9,**99**	648	9,11,**99**	727	11,9,**99**	747	9,11,**99**	t/o

Table 2: Results for our model of Tile Packing, which is a minimisation problem. In the 'time' column, if the reported time is less than the time-out (60,000 milliseconds here), then the reported objective value in the 'obj' column was *proven* optimal; else the time-out is indicated by 't/o' and the reported objective value is either the best value found, but *not* proven optimal, before timing out, or '–', indicating that no feasible solution was found before timing out. Boldface indicates the best performance (objective value or time) on each row.