Comp3620/Comp6320 Artificial Intelligence
Assignment 2: Constraint Programming

April 17, 2017

In this assignment, you will solve logistics problems using the constraint programming tool *MiniZinc*, see `http://www.minizinc.org/`.

The Logistics Problem

Imagine a transport company which needs to send goods by road from a depot to a number of customers. They have a fleet of trucks, of various sizes, some refrigerated and some not. The customers are far away (hundreds of kilometres) so in one round of deliveries, each truck makes at most one journey out from the depot, serving one or more customers, and returning to the depot. It is not necessary that all of the trucks be used for this—in fact, usually it pays to use a fairly small subset of the available trucks. Trucks are expensive things to run, so it is important to allocate the jobs to trucks in a way that reduces the overall cost to the company: as you will see, a good solution to this problem can save thousands of dollars in a single round of deliveries.

We consider goods of two sorts: chilled and ambient. Chilled goods can only be transported in refrigerated trucks. Refrigerated trucks can also carry ambient-temperature goods, but they are more expensive to run than non-refrigerated ones. In the version of the problem considered here, the number of customers is quite small, but some of them have placed orders for more goods than will fit in one truck, so they must be visited by more than one of the trucks. The goods are packed into containers, and each truck can carry a few of these units of goods: larger trucks have greater capacity, but again they are more expensive to run than the small ones.

In the setup of each problem instance, we know:

- Which trucks are available, how many units of goods they can carry, whether they are refrigerated or not, and the cost per kilometre of running them;
- Which customers are to be served, and how many units of each goods type (chilled and ambient) they have ordered;
- The distance by road between each pair of customers, and between each customer and the depot. In some cases, the distance from A to B may not exactly equal the distance from B to A because of (unspecified) details of the road network.

A solution to the logistics problem must specify for each truck the sequence of places it visits, starting and finishing at the depot, and how many units of each of the two goods types it delivers to each customer it visits. Of course, no truck may carry more than its capacity, and all customers get exactly what they ordered. A truck that is not used in this particular delivery round just stays at the depot, thus making a journey of length zero. The cost of each truck's journey is the total distance (in kilometres) it travels multiplied by the cost per kilometre, and the total cost of the operation is the sum of the journey costs for all trucks in the fleet. An optimal solution is one that minimises this cost.

Minizinc

MiniZinc is not a programming language, but a language for specifying constraint satisfaction problems, based on first order logic with a lot of built-in features and support for arithmetic. It comes with back-end software which compiles the problem description into a low-level CSP which can be solved by a conventional constraint-based reasoning system. This may be a finite domain solver, but it could be something else such as a SAT solver or a mathematical programming tool. A central design feature of *MiniZinc* is that the logical definition of the problem should be clearly separated from the program that solves it, so *MiniZinc* itself encourages you to write solver-independent problem specifications, making it very suitable for our purposes.

Typically, the problem is specified in a file (with filename extension `.mzn`) in a way that enables different instances of the problem to be encoded by supplying different data values. *MiniZinc* data files have filename extension `.dzn` and will fix the values of parameters such as the number of trucks, the distances between places and the orders placed by customers. This simple organisation of files enables all of the logistics problems we consider here to be specified without changing the `.mzn` file at all.

For this assignment, you will need to design a logical description of the logistics problem, consistent with the format for input of data and output of solutions described below. This means you must write the constraints, and define the vocabulary they need. You must then run *MiniZinc* with your files as input, and obtain solutions to the various problem instances.

Lab: Learning Minizinc

We'll cover a fair number of the practical details of learning *MiniZinc* and getting *MiniZinc* to work in the lab before your assignment is due. Go to the `Lab_Exercises` directory and open the `Lab_CP_students.pdf` file to see the exercises. They will guide you through a series of steps relevant to the techniques you will need for this assignment.

The most helpful source of more comprehensive information about using *MiniZinc* is probably the tutorial by Marriott and Stuckey, which is also included in the same directory. The material most useful for this assignment is in sections 1 through 4—mostly in the part to the end of section 3.4, in fact. In section 4, only the basics of using predicates and the `alldifferent` constraint are likely to be relevant.

For those who want the technical language spec, that's also available online: `http://www.minizinc.org/doc-lib/minizinc-spec.pdf`; for the rest of us, that's not really necessary.

Assignment Specifics

Once you've completed the above lab exercises, you should be ready to start the assignment. The assignment consists of a series of 4 steps each building on the previous one. These steps are to be completed within directories Q1 to Q4, and are designed to progressively take you to the point where you have a complete *MiniZinc* encoding of this problem and use it within a large neighborhood search (LNS).

For all problems, you will use the Finite Domain (FD) solver `g12-fd`. This is the default solver which is called when running `minizinc` from the command line. However, if you use the *MiniZinc* IDE, you may need to select this solver explicitly, as sometimes the IDE uses the `gecode` solver by default. However, you are free to experiment with other solvers and report some of your results in the `.txt` files mentioned below if you find they bring huge benefits in comparison to the FD solver for your particular encoding.

Your grade will depend primarily on the correctness of your solution/encoding but also on its elegance and efficiency, as well as your creativity (in particular for LNS), and the quality of your experimental observations.

Question 1: Allocating Deliveries to Trucks (30pts)

Problem description

Given the data on the capacities of trucks, and whether they are refrigerated or not, and given the orders placed by some customers, find an allocation of goods to trucks which satisfies all customers orders without exceeding the capacity of any truck and without allocating chilled goods to any non-refrigerated truck.

At this point, we are not concerned with the costs nor with the sequence in which the trucks will visit customers. We also aren't optimising anything, hence any solution satisfying the given constraints will do.

Files we give you

In the directory Q1 you will find:

- `Logistics-Q1.mzn`: the start of the *MiniZinc* encoding, merely declaring the parameters and types required to read the data for this question.
- `Logistics-Q1-4-3.dzn`, `Logistics-Q1-6-3.dzn`, `Logistics-Q1-12-4.dzn`, `Logistics-Q1-21-8.dzn`: 4 data files corresponding to 4 problem instances of increasing difficulty. `Logistics-Q1-`x`-`y`.dzn` has x trucks and y customers.

What you must produce

- In `Logistics-Q1.mzn`: add the *MiniZinc* variable declarations and constraints needed to solve the problem for Q1, and a *MiniZinc* output item outputting the solution in the format described below. Make sure you comment your file, as it will help us understand your intent, especially in case your solution is incorrect overall.
- 4 files `solution-Q1-4-3.csv`, `solution-Q1-6-3.csv`, `solution-Q1-12-4.csv`, `solution-Q1-21-8.csv`: each of these files should contain the solution (in the output format below) found by *MiniZinc* for the respective problem instance. **If you can't solve one of the instances in the order of 1 minute**[1], please create the file but leave the content empty. Note that your encoding may not be able to solve the largest instance with the FD solver.
- 1 file `report-Q1.txt`: telling us, for each problem instance, whether it was solved at all and how long it took to solve it. This is your chance to let us know about any problem you could solve but in times that significantly exceed a minute, or about any substantially better results obtained with a different solver. There is no precise format for this file. You might also want to report about other things that aren't evident from the final code or results, including any other encoding or technique your tried that didn't go so well. Please be reasonably concise.

Solution output format

For Q1, the solution output format is as follows:

```
nb_trucks, nb_customers (not including the depot)
[truck_id, customer_id, chilled_goods_units_delivered, ambient_goods_units_delivered]*
```

Hence the first line specifies the number of trucks and customers in the instance. Each other line specifies for a truck and a customer that trucks delivers to, how many units of chilled goods and how many units of ambient goods that truck delivers to the customer. If the truck doesn't deliver anything to a customer, no corresponding line should be created. Here is an example of the beginning of a solution file for `solution-Q1-6-3.csv`:

[1] 1 min 10 secs is OK, 2 min is pushing it, and 5 min is not OK.

```
6,3
1,1,0,4
3,3,7,0
4,2,4,0
4,3,3,0
etc
```

Please remove the `---------` and `==========` printed by *MiniZinc*. This will automatically happen if you call *MiniZinc* as follows:

```
minizinc Logistics-Q1.mzn Logistics-Q1-6-3.dzn --soln-sep "" --search-complete-msg "" > solution-Q1-6-3.csv
```

MiniZinc has a rather unconventional way of specifying formatted output. To help you produce the required format, we stronly suggest that you have a look at some of the output examples in `minizinc-tute.pdf`, especially those including the use of list comprehensions and the `fix` function. This function turns a variable whose value has been determined (fixed) by the search, into a constant with this value. See in particular p37-38. This is likely to be useful ...

Question 2: Routing the Trucks (30pts)

Problem description

This is probably the trickest part of the assignment. The problem is now extended to include information about the sequence of customers visited by each truck. Given the same data as for Question 1, find not only the allocation of goods to trucks but also for each truck a sequence of places to be visited, starting and finishing at the depot and in between visiting the customers to whom it makes deliveries. This sequence should not visit the same customer twice, and it should not visit a customer to whom the truck does not deliver anything.

Files we give you

In the directory Q2 you will find:

- `Logistics-Q2.mzn`: the start of the *MiniZinc* encoding, which is as for Q1 plus a new parameter (times) giving the maximum number of time steps in a sequence. This is sufficient to (in the worst case) visit all customers and return to the depot.
- `Logistics-Q2-4-3.dzn`, `Logistics-Q2-6-3.dzn`, `Logistics-Q2-12-4.dzn`, `Logistics-Q2-21-8.dzn`: 4 data files which are exactly the same as for Q1.

What you must produce in the directory Q2

- In `Logistics-Q2.mzn`: extend your Q1 encoding by adding the extra *MiniZinc* variable declarations and constraints needed to solve the problem for Q2, and modifying the *MiniZinc* output item to match the format described below.
- 4 files `solution-Q2-4-3.csv`, `solution-Q2-6-3.csv`, `solution-Q2-12-4.csv`, `solution-Q2-21-8.csv`: as before create these files in the Q2 directory and print the solutions to the respective instances you could solve in the order of 1 minute in the format below. Since the problem for Q2 is harder than for Q1, we again don't expect that you'll be able to solve the largest instance.
- 1 file `report-Q2.txt`: telling us, as before, for each problem, whether it was solved and how long it took to solve it. Feel free to include any remark about what worked and what didn't, and why.

Solution output format

This is similar to the format for Q1, but with the addition of an extra time index after the truck id to indicate the time step in the sequence (1, 2 ...) at which the customer is visited:

```
nb_trucks, nb_customers (not including the depot)
[truck_id, time_id, customer_id, chilled_goods_units_delivered, ambient_goods_units_delivered]*
```

Hence the first line specifies the number of trucks and customers in the instance as for Q1. Each other line specifies for a truck, the time index in the sequence at which it visits a given customer, the customer, and how many units of chilled goods and how many units of ambient goods that truck delivers to the customer. As before, if the truck doesn't deliver anything to a customer, no corresponding line should be created, and consequently no line should be included for the depot. Here is an example of the beginning of a solution file for `solution-Q2-6-3.csv`:

```
6,3
1,1,1,0,4
3,1,3,7,0
4,1,2,4,0
4,2,3,3,0
etc
```

specifying, for instance, that truck 4 visits customer 2 at time step 1 and customer 3 at time step 2.

Question 3: Optimising (10pts)

Problem description

If you got that far, this question shouldn't delay you too long. This part is as for Question 2, but this time we require a solution that minimises the total cost. This means you must take into account the distance each truck travels, and the cost running each particular truck to compute the total cost of the solution. Both the distances between each pair of customers or the depot (in Km), and the running cost of each truck (in cents per Km) are now given in the data files.

Files we give you

In the directory Q3 you will find:

- `Logistics-Q3.mzn`: the start of the *MiniZinc* encoding, which is as for Q2 but includes new arrays to read the distance and cost data, as well as a new variable `tot-cost` representing the total solution cost to be optimised.
- `Logistics-Q3-4-3.dzn`, `Logistics-Q3-6-3.dzn`, `Logistics-Q3-12-4.dzn`, `Logistics-Q3-21-8.dzn`: 4 data files now including the distance and cost data.

What you must produce

- In `Logistics-Q3.mzn`: extend your Q2 encoding by adding the extra *MiniZinc* variable declarations and constraints needed to solve the problem for Q3, and slightly modifying the *MiniZinc* output item to match the format described below.
- 4 files `solution-Q3-4-3.csv`, `solution-Q3-6-3.csv`, `solution-Q3-12-4.csv`, `solution-Q3-21-8.csv`: as before create these files in the Q3 directory and print the **optimal** solutions to the respective instances in the format below. Note that intermediate solutions printed by *MiniZinc* with the `-a` option are not optimal. **Create an empty file if you couldn't solve the problem optimally in the**

order of 1 minute. Optimising is notoriously more difficult than finding arbitrary solutions, hence we do not expect that your encoding will be able to optimally solve Logistics-12-4 let alone the larger problem with 21 trucks), but would be impressed if it does!

- 1 file `report-Q3.txt`: telling us, as before, for each problem, whether you could solve it optimally at all and how long it took to solve it. Again here this is your chance to mention problems you could solve but using significantly more time than 1 minute, other solvers tried, and anything else you think we should know.

Solution output format

This is the same as for Q2 except that the first line now includes the total cost of the solution in *dollars*.

```
nb_trucks, nb_customers (not including the depot), tot_cost (in dollar)
[truck_id, time_id, customer_id, chilled_goods_units_delivered, ambient_goods_units_delivered]*
```

For example, the first line could be:

```
6,3,22678.70
```

Question 4: Scaling Up (30pts)

Problem description

If you are still with us, this is the fun part. For the first three questions, you were required only to write *MiniZinc* encodings of the problems, and to note the limits on problem size that can be solved optimally, and on problems that can be solved at all, by *MiniZinc*'s solver alone. In order to scale up to somewhat larger problem instances, you will now implement large neighbourhood search (LNS) in Python, using *MiniZinc* with its default FD solver to search each successive neighbourhood. Your LNS will start from a base solution that we provide and will attempt to improve it.

For this question, you have some opportunity to try creative ideas as to how you might define the neighbourhoods. You should also note the different effects of searching neighbourhoods for (locally) optimal solutions and searching them merely for satisfying solutions. How well does your method scale up? How far can you go?

Implementation Hints

There are many ways your Python program could interact with *MiniZinc* to implement LNS, and we do not want to prescribe one; the only constraints we place are on what must be produced. For information, here are three ideas we found useful in our own implementation:

- We sometimes wanted to search neighbourhoods for a (locally) optimal solution, and sometimes for just any solution. To avoid duplicating the *MiniZinc* model, we removed the 'solve' item from the file `Logistics-Q4.mzn`, and created two small files `Logistics-sat.mzn` or `Logistics-opt.mzn`, each of which consists of just two lines, one being the instruction to include `Logistics-Q4.mzn` and the other being the relevant 'solve' item (solve satisfy, or solve minimize tot_cost).

- In order to specify which part of the solution to keep and which to remove, to define the neighbourhood, we had our Python program write an additional datafile `keep.dzn` containing that information in the form of array assignments. At each iteration of the LNS, we call *MiniZinc* with 3 parameters: one of the mzn files (`Logistics-sat.mzn` or `Logistics-opt.mzn`), the data file for the original problem (for instance `Logistics-Q4-12-4.dzn`), and `keep.dzn`.

- To assign values to part of an array, leaving another part blank, you can use the anonymous decision variable "_" (the underscore character). For example, given an array saying which of four people like, which, we may write for instance

```
likes = array2d(person,person,
[true, _, false, true,
 false, _, true, true,
 false, _, false, true,
 true, _, true, true ]);
```

  to remove the specification of who likes person 2, while keeping the rest. We found this notation useful for deleting just the parts of the solution we wanted to remove to create a neighbourhood.

Note that depending on the effectiveness of your encoding for Q3, it may be that LNS will not improve much your overall results over vanilla *MiniZinc*, except perhaps on the largest instance (21 trucks). Or it could be that it is able to find better solutions, even on smaller instances (e.g. 6 trucks).

Files we give you

- `Logistics-LNS.py`: a near-empty skeleton of your Python program implementing LNS. This skeleton just enforces that the program takes as input the problem data file (e.g. `Logistics-Q4-12-4.dzn`) and the solution file to improve (e.g. `start-solution-12-4.dzn`).
- `Logistics-Q4-4-3.dzn`, `Logistics-Q4-6-3.dzn`, `Logistics-Q4-12-4.dzn`, `Logistics-Q4-21-8.dzn`: 4 data files describing the problem instances, identical to those in Q3.
- `start-solution-4-3.dzn`, `start-solution-6-3.dzn`, `start-solution-12-4.dzn`, `start-solution-21-8.dzn`: the initial solutions to improve on for the respective problem instances. The format for these files is the solution output format for Q3.
- We also give you our `Logistics-sat.mzn` and `Logistics-opt.mzn` but you do not have to use them.

What you must return

- In `Logistics-LNS.py`, write code implementing LNS. This code must print to standard out the initial solution to improve, and, as they are found by the LNS, any new solution found whose cost is better than the current best solution.
- Any other file that your program needs to be able to run. E.g. our own program needs `Logistics-Q4.mzn`.
- `best-solution-4-3.dzn`, `best-solution-6-3.dzn`, `best-solution-12-4.dzn`, `best-solution-21-8.dzn`: the best solution found by your LNS within 1 minute in the first three cases, and two minutes for `best-solution-21-8.dzn`. Use the same solution output format as for Q3.
- 1 file `report-Q4.txt`: containing more details of how quickly your LNS actually converged to the best solution on the smaller problems, and the results of any other experiments you ran on the large ones. You should say how your neighbourhoods were defined—what you destroyed of each solution and what parts of it you kept—and as before comment on what seems to work well or not so well on this logistics problem.

Solution output format

As stated above, this is the same as for Q3.