Vrije Universiteit Brussel
Declarative Programming Assignment 5 (Individual, Assessed) 2019-20

Delivering Cakes in Washington D.C.

Geraint A. Wiggins

This assignment is intended to test your ability to apply the advanced concepts delivered in the Declarative Programming course. You are asked to design and program a delivery management system for a small artisan bakery working in central Washington DC. You should do so using constraint logic programming and definite clause grammars.

Delivery. The bakery specialises in cakes for events (birthdays, weddings, etc.) and is popular with businesses and institutions in the city because deliveries are conducted by bicycle. Each delivery biker has a bike with a large trailer attached. The trailer is 1m x 1m square, and 50cm high, and it is partitioned into four sections, with fixed dividing walls of the same hight. When the lid is open, the biker can easily reach in and pick up a cake, because there are gaps for fingers in the dividing walls, but the cakes are held fixed when in transit. A minor design fault in the trailer is that, when the lid is open, there is no flat surface on which to rest things - this can be annoying, because the bikers are not allowed to put the cakes or their boxes down on the road or on other outdoor surfaces, for reasons of hygiene, and so the boxes cannot be rearranged once the biker has left the bakery. Thus, the boxes must be loaded in the correct order for delivery. The trailers always start out completely full.

There are some practical issues concerning the bike-based deliveries. First, the cost of the delivery is completely determined by the distance involved: the bikers go at a constant speed through the terrible D.C. traffic. So the bakery wants the shortest possible route for each delivery load, and that also benefits the customer because then the cakes are as fresh as possible. Second, bikes are not allowed on all roads in the USA. For example, roads of type “highway” are not usable by bike, whereas city streets are. (You may need to know that a “4WD” is a four-wheel drive vehicle.)

The bikers must always return to the bakery at the end of the route to unhitch their trailer, and to collect their wages, proportional to the time they have worked.

Cakes. The bakery makes 3 kinds of cake, which can be customised to the needs of the client. They are all the same size, to fit a box 49cm square and 25cm high. The relevant difference is the weight of the cake: sponge cakes (type 1) are delightfully light and fluffy, and require only a transparent plastic box (biodegradable, of course) to support them; mousse cakes (type 2) are heavier and need a medium weight cardboard box; and rich fruit cakes (type 3) are very heavy, needing a solid cardboard box. Heavier boxes can support the weight of the lighter ones, but the converse is not the case.

System requirements. The bakery wants you to build a system that will accept a simple list of customer requirements and addresses, and calculate the shortest route that for a single delivery route, given the above constraints, and the geographical information about Washington, specified in the project files supplied. In the event that all the constraints cannot be met, the system should meet as many of them as possible. The system should also plan the packing of the delivery trailer.

Having planned the route and the trailer packing, the system should print out instructions, first for the trailing packing, in terms of 4 stacks, with the lowest job number first, and then for the biker en route. The trailer packing can be simple lists of numbers, but the bikers need something a bit more human: they have to cope with traffic while paying attention to their instructions. So your system should print out instructions for the biker to follow, for example, something like

> "Start from 164.
> Go along road 164-165.
> At 165 turn left into road 165-170.
> At 170, deliver cake 42.
> At 170 go straight on along road 170-180…"

(Note that these directions do not actually form a possible route in the map - these numbers are for example only. Note also that your list of instructions may turn out to be really quite long.)

The bakery is located at 77.0191W 38.8794N. You can use the nearest map node to the actual location of the delivery addresses to estimate the delivery locations. For the purposes of the calculation, the compass coordinates given in the project files may be considered to be equivalent to distance on the surface of the Earth. If you need to make these into integers, for any reason, you could multiply by 1,000 and take the integer part (so keeping 3 decimal places of information).

Today's orders.

Order number	*Cake Type*	*Delivery Location*
1	1	Smithsonian Institution Offices
2	3	Supreme Court of the United States
3	2	Capitol Skyline hotel
4	3	Ted's Bulletin Restaurant
5	1	The Wharf Marina
6	2	The International Spy Museum
7	1	Wang Accounting Services
8	2	St Dominic Church
9	3	Voice of America Radio Station
10	2	Rayburn House Office Building
11	3	Capitol Hill Club
12	2	Founding Farmers DC Restaurant

Hints. Begin by designing the data structure that will hold your route, including the delivery address nodes, the order number, and the section of the trailer in which the order is stored, and any other information you may need. The route-finding part is equivalent to the Traveling Salesman Problem (see Wikipedia, if you want to know the details), which is NP-hard, so you may want to split it up, first determining an estimated optimal order of delivery, given the requirements of packing the trailer, and then finding the actual route and distance that it costs. You may want to use A* search, which is optimal, for the actual route finding: breadth-first search, as in the previous exercise, is not practical here because the paths are too long. You can look up Algorithm A* on Wikipedia, if you're not familiar with it. A library predicate that is likely to be particularly useful is `keysort/2`. The easiest way to approach the programming is to build the search algorithm and get it to work correctly for a single route between two places. Then you can add constraints to insist on intermediate calling points, in any particular order, as you need to, afterwards.

Submission. You should submit your work via Canvas before the deadline specified in the Canvas Assignment module. You should submit **one single .pl file**. Your file should include, in comments, at least one example of a correct output route and trailer loading plan. You may (indeed, you have to) use libraries (see the SWIPL website for documentation) but should submit only the code that you wrote. You may use any code from the model solution to Assignment 3 that you think is useful, but you should acknowledge this in comments if you do so.

Please remember that you MAY NOT SHARE CODE and that all code you submit, other than anything you borrow from the Assignment 3 solution, must be YOUR OWN CODE. If you submit code that is not your own, and you are caught, you will score zero for the whole assignment.