Declarative Programming Report

Hereman Nicolas

January 2017

1 Description of the different approaches

1.1 is_valid(+P)

The predicate *is_valid/1* checks if the plan respects the format and all the hard constraints :

1. All the vehicles are scheduled exactly once.
2. The orders are shipped at most once.
3. Vehicles start at the beginning of the working day and do not drive after after the end.
4. Vehicles do not carry products from depot to depot.
5. Vehicles only take orders in a depot if there is enough product.
6. Vehicles do not carry more products that their capacity allow them.

1.2 profit(+P,-Profit)

This predicate *profit/2* computes the Profit of plan P by computing the revenue and the expenses.

The revenue is the sum of the revenue of each schedule. The order of a schedule is computed with the predicate earning/3 applied to all the orders delivered.

The expenses are the sum of all the expenses induced by each schedule. The expenses of a schedule is computed by computing the driving duration of his route and use it in the formula *Distance * KmCost + UsageCost*.

1.3 find_optimal(-P)

The predicate *find_optimal/1* finds a plan P which maximizes the profit.

It makes use of dynamic predicate *best/2* with *assert/1* and *retract/1* to keep trace of the best solution.

1.4 find_heuristically(-P)

The predicate *find_heuristically/1* generate a valid plan P approximately maximizing the profit.

The idea is to generate a schedule for every vehicle every day. In order to do this, we use a list of *DepotId/Inventory* to keep track of the inventories and a list of orders not delivered yet. The schedules are generated day by day.

To generate the route of a schedule, we get all the list of remaining orders from size 0 to 2 which can be picked up at the last depot and carried by the vehicle. We take the one which gives the best profit under the condition that it remains enough time to reach a depot. These orders are added to the route. The reachable depot which gives the most choice in orders is added to the route. Then we do it again until we can not reach a depot before the end of the day.

1.5 pretty_print(+P)

The predicate *pretty_print(+P)* print the plan in a nicely readable way. It goes through each schedule and print all the action done during the route. It is the same print as the one in the slides.

1.6 Extended Functionality

The predicates *is_valid(?P)* and *is_optimal(?P)* were not implemented unfortunately.

2 Strengths and weaknesses

2.1 Non-working predicates

Unfortunately, the predicate *find_optimal/1* doesn't work. It makes use of *is_valid(?P)* which is not implemented.

The predicates *update_inventory/4* has one problem. In the case where all the amount of product "px" is taken it gives both solution : without "px" and with "px/0". This means that when a cut is needed to block useless backtracking, it is used after *update_inventory/4*. Otherwise, the predicate works perfectly.

All the predicates except these two are working perfectly.

2.2 Quality of heuristic

The heuristic find a solution for every example but the result is far from the optimal. The profit found are positive except for two instance where the profit is zero.

2.3 Non-Functional Requirements

2.3.1 Test in computer rooms

The code couldn't be tested on the computer rooms but there is not any reason for it to not work. It was tested on multiple version of SWI-prolog to make sure this kind of problem will not happen.

2.3.2 Generality

The code does not use any property of the instances so it should be as general as possible.

2.3.3 Procedural style

if-statements were avoided in the code. Asserts is only used for *find_optimal/1*. Cuts are used multiple time but it was to avoid unnecessary backtracking.

2.3.4 Modularity

The auxiliary predicates share a same module. Each predicate of the core has his own module.

The module utility define some predicates use-full in the different core predicates. The module delivery import all the others.

2.3.5 Efficiency

All the predicates run in under 2 minutes on my laptop. It should be the same on the lab computers.

2.4 Experimental Results

2.4.1 find_optimal(-P)

As said in 2.1, the predicate *find_optimal/1* need *is_valid(?P)* to work. So there is no experimental results available.

2.4.2 find_heuristically(-P)

Here are the results of *find_heuristically/1* on small and large instances :

Instance	Profit ($)	Time (s)
single_small	785	0.026
multi_depots_small	798	0.014
multi_vehicles_small	796.4	0.03
multi_days_small	640	0.024
multi_small	206.6	0.013
single_large	3168.8	10.631
multi_depots_large	0	0
multi_vehicles_large	3526.6	25.043
multi_days_large	3189	26.768
multi_large	0	0.004