CP(M) Assessed Exercise 2

Tom Wallis, Matric: 2025138

November 2015

1 Description of Model

The model used represents the problem as an array who's length is its number of meetings. The values of each element of the matrix is the timeslot of the meeting the element represents. This model was sufficient enough to complete the Solve program provided, through implementing simple Distance constraints on the timeslots to ensure that travel time (and meeting duration) were honoured in the solution.

Not only does this allow for posting simple constraints to represent the problem, but constraining the timeslots for optimisation is easy. One need only place a constraint on each member of the array which states that the member should be less than some bound, which is also an IntVar within the solver. With each member bounded, one can place a minimisation constraint on the bound, ensuring that we use the fewest timeslots possible and that the times allocated to each meeting follows this bound. This, together with a Choco call for limiting the time spent finding the solution, was all that was required to modify Solve to make Optimize.

An optimisation to this model to prevent duplicates was to memoize the constraints posted, by taking note in a binary matrix. Only two meetings attended by at least one person in common should be constrained (and this is how the constraints are posted), but in addition, only one constraint should be posted if two of the same people attend both meetings. Therefore, for every constraint posted, note was taken of the constraint so that it wouldn't come up a second time.

2 Description of Heuristics

Two heuristics were chosen to attempt to speed the model up: *Fail First* and *Domain over Weighted Degree*.

Domain over Weighted Degree was chosen because of its general effectiveness in a variety of problems, and its adaptive nature where variables that consistently produce domain wipeouts have weights that change over time. In context, this is akin to choosing to schedule meetings that have a low chance of causing backtracking first, and choosing harder meetings to schedule later.

Fail First was chosen because it behaves potentially quite differently, as it chooses the most difficult-to-schedule variables first. The notion behind testing this heuristic was that, because it comes at the problem of variable ordering from a different angle, it might yield surprising or enlightening results. Fail First is equivalent to attempting to fit the most difficult meetings (or meetings with the most "constrainedness") first, and allowing the more forgiving, easier-to-schedule meetings to find some solution after the harder part was solved.

It was suspected that Fail First would prove to be the more effective heuristic in this case, as the harder-to-schedule meetings would be instantiated without any constraint on their domain, and easy-to-schedule meetings would quickly be instantiated afterward, leading to a solution.

3 Computational Study

The code appears to run rather quickly. Particularly, making use of Choco's built-in Distance constraint is quicker than constructing one's own out of Arithmetic and Or constraints.

Fortunately, the additional constraints in Optimize do not appear to slow the code down, though each member of the meetings array has an additional constraint on it. For larger problems, this additional constraint is a small proportion of the overall number of constraints, so its affect is assumed to be more minimal as complexity increases.

For the *Domain over Weighted Degree* heuristic:

Problem Number	Nodes	Time (ms)
1	25	46.92
2	131	47.03
3	116	44.07
4	60	32.14
5	84	38.31
6	397	123.25
7	917	127.72
8	1323	139.18
9	451	80.96
10	405	75.64

For the *Fail First* heuristic:

Problem Number	Nodes	Time (ms)
1	133	33.73
2	891	90.07
3	29	16.51
4	92	23.33
5	57	21.21
6	712	94.61
7	1317	144.71
8	46655	1221.63
9	1010	111.79
10	574	72.14

The default variable and value picking heuristic Choco provides was also tested:

Problem Number	Nodes	Time (ms)
1	21	20.15
2	21	21.87
3	21	21.25
4	21	18.90
5	21	18.45
6	21	20.21
7	21	27.54
8	21	18.31
9	21	21.38
10	21	19.63

To great surprise, the built-in heuristic vastly outperformed any of the alternative heuristics. It seems that Domain over Weighted Degree performed slightly better in this limited sample of problems. Over a larger problem set, or over a problem set of more varied examples, this may change. However, initial results seem to favour Domain over Weighted Degree in this case.
However, neither heuristic could out-perform the built-in Choco heuristic. Therefore, while the method to construct the Domain over Weighted Degree heuristic will be included in the submitted code, no calls to post the heuristic will be made, and the compiled code will use the built-in heuristic.

4 Consideration of Alternative Model

An alternative model to consider would be to make use of Interval Graphs. Edges in the graph have a weight corresponding to the time taken for an agent to get between the two points, and constraints should be posted for each one. While the constraints posted would be similar to the model implemented, constructing and processing the graph as a model may allow for alternative heuristics and speedups associated with similar problems.

Another model worth considering would be to draw parallels to graph colouring, where each timeslot is a colour. Graph colouring is a well studied problem with clever heuristics and well-understood properties, but care should be taken to avoid certain problems. For instance, associating colours with timeslots is tempting, but colours which go unused may need to be kept track of, as they may be times all agents travel from one meeting to another and therefore contribute to the overall number of timeslots needed for a solution.