Report on the implementation of the Aircraft Landing Problem

Modeling and Solving Constrained Optimization Problems

Bernhard Mallinger
0707663

December 30, 2012

1 The problem

This exercise deals with implementing the Aircraft Landing Problem, which is specified by the following constraints:

TIME Each aircraft has a minium, preferred and maximum time of landing.

COST There are possibly different costs for airplanes being too late or too early.

RUNWAY Runways can be unavailable for certain periods.

SEQUENCE DELAY Sequences of airplane landings have to keep a certain offset depending on the airplane types.

TYPE SEQUENCE There must not be more than 3 landings of the same airplane type in sequence and in any sequence of 5 landings, there must be at most 2 medium sized and 1 large airplane.

LANDINGS 30 MINS There is a limit to the number of landings that can take place in 30 minutes.

2 Getting acquainted with the problem, modeling and Gecode

Due to my lack of major experience in any of the three mentioned fields, I decided to start by relatively freely thinking about and trying different possible ways of modeling. This phase

should not have the concrete demand of yielding useable solutions but to produce experience and generate a feel as to which modeling decisions actually lead to correct results and furthermore which of the ones that are correct are computationally feasible.

TIME The first decision was to represent the times when aircrafts land as array of concrete periods `aircraftTimes` of the length of the number of aircrafts. Each entry corresponds to one aircraft. This gives a natural way of describing the time frame, in which it is possible for them to land, by specifying the domain for each entry to be this time as given by the instance.

COST This representation also allows to model the cost function quite easily. Since being late and early is punished by different costs, it is necessary to calculate their values separately. The amount of for instance delay for an aircraft `i` is given by `max(0, aircraftTimes[i] - instance.aircraft[i].preferredTime)`. Using this, an array of delays can be constructed, which is suitable for use with the `linear`-constraint in combination with the cost vector. The resulting value summed up with an analoguously constructed value for being too early make up the cost value.

LANDINGS 30 MINS In order to represent the number of landings within a certain time frame, a different representation has to be found. As this constraint concerns ranges of periods rather than aircrafts, I employed a set array `timeAircrafts`, whose entries represent the set of airplanes landing at the respective periods. It is linked to `aircraftTimes` by requiring that the index of the airplane `i` is an element of `timeAircrafts[ aircraftTimes[ i ] ]`. Now it's only necessary to sum up the cardinalities of these sets for each time frame of 30 minutes and stating that this sum has to be smaller than the number given by the instance.

Since this simple constraint, if it is tight enough to actually matter, already leads to very bad runtimes for small instances (20-30 planes), I decided to try to vary it in order to allow for better propagation. In the variation, the union of all sets of airplanes for each 30-minute-frame is calculated in a set variable which whose maximal cardinality is bounded by the limit given by the instance. As this change does not change the semantics, the number of solutions and failed nodes stays the same, but depending on the instance, the runtime is sometimes noticably affected with some instances being solved nearly twice as fast whereas others show no significant change. This suggests that the internal propagation mechanism of Gecode can in some cases deal more efficiently with the second variation.

Branching Based on these constraints, it is possible to do investigations on the branching parameters. For branching, strategies can be selected for which variable to branch on as well as which values of the domain to pick. It is easily empirically evident that these parameters change the required computation time by multiple orders of magnitude and decide whether a formulation finishes within instant or is computaitonally infeasible.

Due to the size of the search space, comprehensive measurements and tests with larger instances or selectors, that seem less promising, would exceed the scope of this assignment. Therefore I'm focussing on managable instance sizes and take a closer look at certain cases.

Instances with no severe constaint are usually solved quickly using the `INT_VAR_NONE`-selector. However on instances that are already tightly constrainted by the **LANDINGS 30 MINS**-constraint, the `INT_VAR_AFC_MAX`- along with the `INT_VAR_SIZE_MIN`-selector has shown to be exceptionally efficient. They have in common that they provoke quick failures, thereby cutting the search space. Other selectors preferring variables with large domains lead to disastrous runtimes up to the point where it makes no sense to include them in this investigation.

Figure **??** shows measurements using different combination of promising selectors. As not many constraints have been posted so far, ordering based on the degree of variables somewhat corresponds to having no special variable selection order. Therefore other selectors which are geared towards quick failures and are based on the node degree don't have any effect in this case.

Since preferred values for landing times of common instances are somewhere between the minima and maxima, choosing values around the extremes of the domain is not optimal as the direct comparison of `INT_VAL_MAX` and `INT_VAL_MED` shows, where the latter is faster by a factor of roughly 2.

8 aircrafts,	`VAR_AFC_MAX`		`VAR_SIZE_MIN`		`VAR_NONE`	
3 per 30 mins	time	nodes	time	nodes	time	nodes
`VAL_MAX`	0.78	3054	0.53	2141	6.80	51245
`VAL_MED`	0.32	1807	0.18	563	9.74	74163
`VAL_SPLIT_MAX`	1.75	12361	0.57	2376	7.31	59378

Figure 1: Measurements of the run time behaviour on an instance containing 8 airplanes scheduled in 120 periods with a maximum of 3 per 30 minutes. The time is given in seconds, "node" means the number of nodes expanded during the search.