Constraint Programming AX2 Report

Michael Kilian - 1003819k

November 22, 2013

1 Heuristics Used

The following variable/value ordering heuristics were tried:

- Variable Orderings
  - Default: By default Choco chooses the variable with the smallest current domain. This is included for comparison.
  - Static: Assigned the variables in the order $s_1, s_2, ..., s_n$.
  - Middle-In: Assign the variables in the order s_1, s_n, s_2, s_{n-1} ,etc. This heuristic was proposed by Barbara Smith [3, 2]
  - Middle-Out: starting at the middle of the schedule variables are assigned outwards towards the edges.
- Value Orderings
  - Increasing: the default used by Choco. This picks values from the domain sequentially from lowest to highest.
  - Most Expensive Scene First: This chooses the scene in domain of the current slot to be assigned which is the most expensive, where expense is the sum cost of all the actors participating in the scene times the duration of the scene

Note that the Most Expensive Scene First is combined with a static variable ordering. The rationale is that by scheduling scenes sequentially by order of expense, we increase the chances of scheduling the most expensive actors in consecutive scenes. An expensive scene implies either a large number of actors, high cost actors, or both. Essentially then by scheduling these first we are using a fail-first approach.

2 Empirical Evaluation

2.1 Basic Comparison of Performance

To try and assess the basic effectiveness of each heuristic, each was tried on a subset of the data provided. For each problem, the maximum allowed cost was chosen so that the problem can be solved non-trivially. A value was found by first finding a solution trivially then setting the parameter below that so as to keep the problem solvable but reasonably challenging. The data sets chosen and the style of this evaluation were inspired by Barbara Smith's exploration of caching on the scheduling problem[2].

Table 1 summarises the results. The instances are displayed in increasing size of schedule moving down the table. Note a '-' denotes non-completion (the test had to be abandoned as it was taking too long). The maximum allowed hold day cost parameter given to each instance is the number in brackets next to the instance name. Each test was ran three times and the minimum value of the three runs was taken.

These results show that, in general, static variable ordering with most expensive scene value selection is the most effective of the heuristics tested. There are however some anomalous cases which should be highlighted. Firstly we can see that the Middle Out heuristic performed unusually well on the film119 instance. Likewise the Static variable ordering completed the MobStory instance blindingly fast compared to the other heuristics. Recognising exactly why these occurred would require an extensive analysis of execution plan over the instance. However it was clear from the experimentation that these performance values did not hold as the maximum hold cost was tightened: tightening film119 significantly increased the run time of Middle Out and tightening MobStory by 1 resulted in a non-completion with the Static ordering.

	Time (ms)				
	Static	Middling In	Middle Out	Static W/ Most Expensive	Default
tiny01 (1)	144	371	568	630	399
small00 (12)	1455	1126	589	349	1214
concert (20)	23,321	20,156	1984	1018	19,057
film12 (100)	81,175	897,004	1,044,595	46,913	-
film119 (500)	1,678,066	995,080	35,119	3298	-
film117 (365)	-	-	-	25,485	-
MobStory (500)	9611	-	-	143,294	-

Table 1: Comparison of Runtime on Heuristics

	Maximum Allowed Cost										
	20	18	16	15	14	13	12	11	10	8	5
solvable?	t	t	t	t	t	t	t	f	f	f	f
time(ms)	237	326	414	400	388	411	355	1864	1784	806	950
nodes	5	6	9	10	9	8	8	166	152	113	47

Table 2: Examination of small00.dat

2.2 Investigating Tightening of the Maximum Cost

Using the Static with Most Expensive heuristic, the effect of tightening the acceptable hold day cost was investigated for three problems. Sadly the larger problems such as MobStory could not be tackled due to the run-time required to solve and instance. Tables 2,3 & 4 show the results.

With small00, there is a clear spike in complexity, shown by the increased run-time and node usage, at the transition from solvable to unsolvable. However as the schedule becomes more and more constrained, the complexity slowly drops. This is in line with what we might expect (due to phase transition). It is worth noting though that there is no discernible trend between the complexity of the solvable instances. The minimum possible hold day cost for small00 is 12.

Table 3 shows a clear example of a phase transition on the concert problem. The complexity of the solvable instances rises as we approach the transition from true to false, with the complexity peaking when the minimum allowed cost is 18. This continues through to the false instances and then the complexity gradually reduces as we further tighten the schedule. The minimum possible hold day cost for the problem is 17.

The results for film12 are somewhat harder to draw conclusions from, especially since some of the analysis could not be completed due to extreme run times. However there is still evidence of a peak in complexity as we approach the phase transition. The minimum possible old day cost is somewhere between 60 and 65.

3 Promising Heuristics

There are several other heuristics and optimisations which could be promising.
Firstly I would highlight the Actors Equivalent and Pairwise Subsumption techniques [1]. Significant time was put into trying to implement both of these methods as ValSelectors and then by posting constraints; I feel I came very close. However attempts at both of these were plagued by the same problems:

1. From a ValSelector, there is no way (that I can find) to tell what time slot was last assigned efficiently. The closest I could find was to loop through the decision variables until one is found that has not been instantiated. Note this implies a static variable ordering would be required.

2. It is not possible to recover the actors present at a given time from the solver. In attempting these a created a dual variable matrix actorsPresentAtTime which was effectively the transpose of the present matrix (ie for each time i, actor j, if actor j is present at time i then actorsPresentAtTime[i][j] == 1). While this in theory could effectively let one look up who is present at a given time, Choco would not allow me to retrieve the value of this array

In hindsight, the correct way to approach this may be to assigned a custom branching strategy. However I could not figure out how to do this in Choco in time to test it.
Another very promising strategy is the caching strategy described by Barbara Smith [2]. Again this involves reading the state of the current solution and altering the branching strategy of the solver.

	Maximum Allowed Cost										
	20	19	18	17	16	15	14	13	11	8	5
solvable?	t	t	t	t	f	f	f	f	f	f	f
time(ms)	788	4217	37,829	31,546	31,909	30,466	24,366	20,925	15,206	8726	4592
nodes	18	836	9378	7932	8144	6690	5437	4130	2500	1109	373

Table 3: Examination of concert.dat

	Maximum Allowed Cost										
	160	140	120	100	80	70	65	60	55	20	0
solvable?	t	t	t	t	t	t	t	f	-	f	f
time(ms)	55,349	68,636	44,843	57,683	809,031	40,201,520	3,482,440	41,790,948	-	869,149	15,376
nodes	3237	7474	4146	4844	57,038	228,407		374,179	124,936	25,916	206

Table 4: Examination of concert.dat

References

[1] Peter J. Stuckey Maria Garcia de la Banda and Geoffrey Chu. Solving Talent Scheduling with Dynamic Programming. *INFORMS Journal on Computing*, 23(1):120–137, 2011.

[2] Barbara M. Smith. Caching Search States In Permutation Problems, 2005.

[3] Barbara M. Smith. Constraint Programming in Practice: Scheduling a Rehearsal. *Available online from http://www.dcs.st-and.ac.uk/ apes*, 2033.