1 Using Control Knowledge in SAT

In Exercise 10, we are required to design some domain-specific constraints for the Logistic domain in attempt to solve larger instances. A trivial example is given for the Blocksworld. However, this is not rather inspiring, because we cannot always find such easy plans for a complex domain.

Firstly, I'm trying to extract rules that sound reasonable, but are not derived by the basic encoding. That is, I believe there might be some common knowledge to do something but failing to do it does not violate any logical laws. Note that the goal only involves fluents with the form (at ?package ?loc). A trivial idea immediately occurs to me (which is also the hint in handout): Since transiting one package is totally independent with transiting another, there is no need to move a package again once it arrives at its goal location.

$$\text{(at package package's-goal-loc)@}t \rightarrow \text{(at package package's-goal-loc)@}t{+}1 \tag{1}$$

This control knowledge is not adequately efficient, but it maintains optimization. So we still always can find a plan as long as the problem is solvable.

Moreover, I notice that the requirement says trucks can only travel around the same city. So we could limit the load-truck times for each package. My constraint is that

$$\text{(load-truck package truck loc)@}t \rightarrow \text{(at-package-loc)@}0 \vee \text{(same-city loc package's goal loc)@}0 \tag{2}$$

where "same-city" is defined as

$$\text{Predicate (same-city ?loc1 ?loc2): loc1 and loc2 are in the same city} \tag{3}$$

It can be implemented by

$$\begin{gathered}\text{(same-city loc1 loc2)@}0 \leftrightarrow \text{loc1.name and loc2.name have the same prefix} \\ \text{for example, 'city2-1' and 'city2-4'}\end{gathered} \tag{4}$$

Under such condition, a package can be loaded on a truck at most twice. More specifically, one is as its first move when it remains the initial location; the other is to travel towards the goal as its last motion. Otherwise, it is prohibited to get on a truck.

To understand this rule, take a thought that except traveling from the initial to the nearest airport (then fly to the goal's city) and from the airport of goal's city to the goal location, any other truck trip is not a must.

So till now, I strongly preserve the completeness when plans exist. And quite hopefully, the control knowledge above might also maintain the optimality, which I have not tried to prove yet. However, beating around the optimization does not improve efficiency very much. Then I come up with another rule—A plane cannot land twice at the same airport. Define

$$\text{Predicate (plane-visited-airport ?plane ?airport): plane has ever landed at airport} \tag{5}$$

Mathematically,

$$\begin{aligned}&\text{(plane-visited-airport plane airport)@}t{+}1 \leftrightarrow \\ &\qquad \text{(plane-visited-airport plane airport)@}t \vee \text{(at plane airport)@}t{+}1\end{aligned} \tag{6}$$

Then add corresponding constraints:

$$\neg\text{(plane-visited-airport plane airport)@}0 \tag{7}$$

i.e. initial location is not regarded as 'visited'

$$\text{(plane-visited-airport plane airport)@}t \rightarrow \neg\text{(fly-airplane plane any-airport airport)@}t{+}1 \tag{8}$$

With these three control knowledge, I made a test and comparison (Table 1) on *problem01.pddl*. The query strategy is set as "3:11:2 -q ramp", and other options remain default. As a supplement , I also tried on *problem02.pddl* with query "7 -q fixed".

Problem	Encoding	Plangraph	Total time (s)	Actions
01	basic	false	47.74	46
01	basic	true	46.66	44
01	logistics	false	42.65	35
01	logistics	true	45.01	35
02	basic	true	488.91	52
02	logistics	false	473.26	36

In conclusion, my control knowledge works better than plangraph, although it does not accelerate evidently (takes too much time to solve *problem02.pddl*).

2 Understanding Planning Problems

Run *run_experiments.py*, we will obtain statistics as below (Table 2, 3, 4). In the names of column, the letter after underline specifies the plangraph configerations:

n : -p false

f : -p true -l fmutex

r : -p true -l reachable

b : -p true -l both

The query strategy is "1:30:1 -q ramp", which guarantees the solution is optimal with respect to Step. And this strategy also accounts for why different procedures stop at the same step (except for timeout) in the same problem. Such a query approach is not always best certainly. Nevertheless, it is a very good method in these tasks because the solution steps are quite small in most cases. We could apply or come up with other quick strategies such as dichotomy, geometric run-time bounding etc., but there is no need to do that. To implement those methods is far more troublesome, but pays off little. And probably, our primary strategy might perform better when an optimal (w.r.t step) plan is required.

Miconic

Problem	Time_n	Actions_n	Time_f	Actions_f	Time_r	Actions_r	Time_b	Actions_b	Step
01	0.27	4	0.27	4	0.27	4	0.27	4	4
02	0.32	7	0.34	7	0.33	7	0.33	4	6
03	0.45	10	0.44	10	0.47	10	0.44	10	8
04	3.00	14	1.23	14	2.71	14	1.23	14	12
05	18.05	17	3.73	17	16.78	17	3.79	17	14
06	58.19	19	18.78	19	53.68	19	18.60	19	14
07	293.06	23	63.9	23	292.04	23	66.09	23	18
08	-	-	-	-	-	-	-	-	-
09	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-

Pipesworld

Problem	Time_n	Actions_n	Time_f	Actions_f	Time_r	Actions_r	Time_b	Actions_b	Step
01	4.73	5	2.34	5	2.43	5	2.39	5	3
02	8.63	12	4.24	12	4.45	12	4.30	12	6
03	49.88	9	22.06	11	26.89	9	23.35	9	6
04	45.49	11	22.96	11	26.76	12	22.53	11	6
05	-	-	-	-	-	-	-	-	-
06	-	-	-	-	-	-	-	-	-
07	-	-	-	-	-	-	-	-	-
08	-	-	-	-	-	-	-	-	-
09	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-

Rovers

Problem	Time_n	Actions_n	Time_f	Actions_f	Time_r	Actions_r	Time_b	Actions_b	Step
01	1.48	13	1.64	13	1.99	14	1.52	13	8
02	0.74	13	0.86	14	1.05	13	0.77	14	6
03	2.07	16	2.27	20	2.57	18	2.08	21	9
04	1.65	17	1.99	16	2.02	18	1.69	16	6
05	21.60	35	18.85	33	23.15	32	16.62	38	14
06	-	-	104.77	48	-	-	80.12	49	20
07	30.98	37	27.88	35	31.94	31	24.46	35	12
08	-	-	-	-	-	-	-	-	-
09	-	-	-	-	-	-	-	-	-
10	-	-	-	-	-	-	-	-	-

For Miconic domain, reachable action axioms have little effect but fluent mutex improve the planning dramatically. In contrast, reachable action axioms performs well in Pipesworld domain. And at last, neither of the two kinds of constraints accelerate the Rovers problems efficiently.

In fact, after inspecting the CNF files (generated by "-d true"), we find that there are much more reach-type constraints in Pipesworld than those in the other two, and that explains why it becomes efficient, whereas Miconic is exactly the contrary. As for Rovers domain, perhaps the problems are relatively trivial (most of them can be basically solved in 30s), so these axioms make few differences to planning Rovers problems.