Tomáš Balyo (biotomas@gmail.com)
Faculty of Mathematics and Physics,
Charles University in Prague
Czech Republic

Lukáš Chrpa (l.chrpa@hud.ac.uk)
PARK Research Group
School of Computing and Engineering
University of Huddersfield

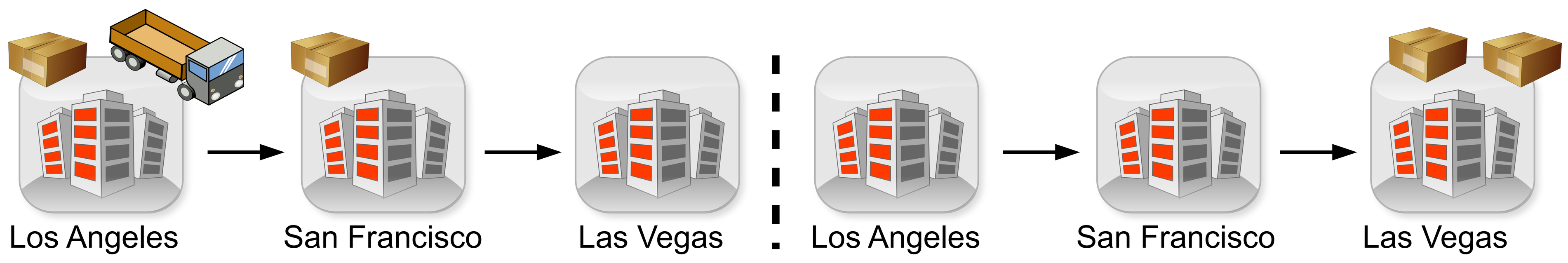
Eliminating All Redundant Actions from Plans Using SAT and MaxSAT

What is Planning?

- World states are described as values of state variables.
- Actions change the state of the world by changing the values of state variables by their effects
- Actions also have preconditions and are applicable only when their preconditions hold in the given state.

Objective: given a set a of actions, an intial world state and the description of a goal state find a valid sequence of actions (a plan), that transforms the world from the initial state to a goal state.

Example: delivering 2 packages to Las Vegas



State Variables and their domains:

- Truck location T, dom(T) = {LA, SF, LV}
- Package locations P and Q dom(P) = dom(Q) = {LA, SF, LV, Tr}

Initial State: T=LA, P=LA, Q=SF
Goal State: P=LV, Q=LV

Actions:

- move(x,y)=[prec: {T=x}, eff: {T=y}]
- loadP(x)=[prec: {T=x, P=x}, eff: {P=Tr}]
- loadQ(x)=[prec: {T=x, Q=x}, eff: {Q=Tr}]
- dropP(x)=[prec: {T=x, P=Tr}, eff: {P=x}]
- dropQ(x)=[prec: {T=x, Q=Tr}, eff: {Q=x}]

Where x,y are LA, SF, and LV

Optimal Plan: loadP(LA), move(LA,SF), loadQ(SF), move(SF,LV), dropP(LV), dropQ(LV)

Redundant Actions and Plans

- Actions that can be removed from a plan without violating its validity are called redundant actions (useless actions).
- Determining whether a plan is redundant, i.e., contains at least one redundant action is NP – complete.
- A plan containing no redundant actions is a Perfectly Justified Plan.

Removing Redundant Actions

- Prior to this work only incomplete polynomial heuristic algorithms
  - Action Elimination (Nakhost, and Müller, 2010)
  - Removing pairs and groups of inverse actions (Chrpa, McCluskey, Osborne, 2012)
  - Neither guarantees removing all redundant actions
- The process of removing redundant action is not confluent, i.e., the result depends on the order in which we eliminate redundant actions
- Achieving perfect justification does not mean, that no better result can be obtained (by eliminating different redundant actions)

Some Redundant Plans

P1: loadP(LA), move(LA,SF), loadQ(SF), move(SF,LV), dropP(LV), dropQ(LV), move(LV,LA)

The first three move actions together are redundant. If we remove them we obtain an optimal plan.

move(LV,LA) is redundant, the goal conditions are already satisfied

P2: move(LA,LV), move(LV,SF), move(SF,LA), loadP(LA), move(LA,SF), loadQ(SF), move(SF,LV), dropP(LV), dropQ(LV)

We can remove either move(LA,LV) + move(LV,LA) or move(LV,LA) + move(LA,SF) + move(SF,LV) to obtain a perfectly justified plan. An optimal plan cannot be obtained.

P3: loadP(LA), move(LA, LV), move(LV,LA), move(LA,SF), move(SF,LV), dropP(LV), move(LV,SF), loadQ(SF), move(SF, LV), dropQ(LV)

Our Approach: Encode Plan Redundancy into SAT

We construct a CNF formula for a problem and plan

- The formula contains Boolean variables a_i representing whether thei-th action of the original plan is required for a reduced plan.
- Each satisfying assignment of the formula represents a valid reduced plan for the given plan and planning problem.
- Using the truth values of a_i we can determine which actions are redundant.

RedundancyElimination (Π, P)
  $F_{\Pi,P} :=$ encodeRedundancy(Π, P)
  while isSatisfiable$(F_{\Pi,P})$ **do**
    $\phi :=$ getSatAssignment$(F_{\Pi,P})$
    $P := P_\phi$
    $F_{\Pi,P} :=$ encodeRedundancy(Π, P)
  return P

IncrementalRedundancyElimination (Π, P)
  solver = **new** SatSolver
  solver.addClauses(encodeRedundancy(Π, P))
  while solver.isSatisfiable() **do**
    $\phi :=$ solver.getSatAssignment()
    $C := \bigvee\{\neg a_i | a_i \in P_\phi\}$
    solver.addClause(C)
    foreach $a_i \in P$ **do if** $\phi(a_i) = False$ **then**
      solver.addClause$(\{\neg a_i\})$
    $P := P_\phi$
  return P

MaximumRedundancyEliminaion (Π, P)
  $F :=$ encodeMaximumRedundancy(Π, P)
  $\phi :=$ partialMaxSatSolver(F)
  return P_ϕ

Using the encoding to eliminate redundancy

- By adding a clause $(\neg a_1 \vee ... \vee \neg a_n)$ to the previous formula we obtain a formula that is satisfiable if and only if the input plan is redundant for the given problem.
- Using this encoding iteratively we can indetify and remove all redundant actions from a plan.
- A more efficient incremental SAT based implementation of the algorithm is also possible.
- To eliminate the maximum number of redundant actions we can use partial MaxSAT solving
  - A partial MaxSAT solver satisfies all hard clauses and as many soft clauses as possible
  - We use our CNF formula as hard clauses and $(\neg a_i)$ as soft clauses (for each action)

Table 1: Experimental results on the plans for the IPC 2011 domains found by the planners Fast Downward, Metric FF, and Madagascar. The planners were run with a time limit of 10 minutes. The column "#Plans" contains the number of plans found and "Length" represents the sum of their lengths. By Δ_{ALG} and T_{ALG} we mean the total number of removed redundant actions and the time in seconds it took for all plans for a given algorithm ALG. The algorithms are Action Elimination (AE), Action Elimination followed by SAT reduction (AE+S), SAT reduction on the original plan (SAT), and maximum elimination using a MaxSat solver (MAX).

	Domain	#Plans	Length	Δ_{AE}	T_{AE}	Δ_{AE+S}	T_{AE+S}	Δ_{SAT}	T_{SAT}	Δ_{MAX}	T_{MAX}
Fast Downward	barman	20	3749	528	0,52	582	3,44	596	7,18	629	0,44
	elevators	20	4625	94	0,84	94	2,41	94	3,45	94	0,19
	floortile	5	234	22	0,06	22	0,20	22	0,27	22	0,00
	nomystery	13	451	0	0,05	0	0,47	0	0,48	0	0,00
	parking	20	1494	4	0,17	4	1,21	4	1,26	4	0,03
	pegsol	20	644	0	0,11	0	1,11	0	1,18	0	0,02
	scanalyzer	20	823	26	0,10	26	1,16	26	1,33	26	0,03
	sokoban	17	5094	244	0,62	458	5,25	458	8,39	460	1,84
	tidybot	16	1046	64	0,14	64	0,91	64	1,28	64	0,03
	transport	17	4059	289	0,65	289	1,64	289	2,93	290	0,20
	visitall	20	28776	122	3,66	122	9,47	122	12,89	122	7,77
	woodworking	20	1605	27	0,41	27	1,16	27	1,33	30	0,03
Madagascar	barman	8	1785	303	0,25	303	1,59	303	3,53	318	0,30
	elevators	20	11122	2848	1,46	3017	4,13	3021	17,62	3138	2,03
	floortile	20	1722	30	0,39	30	1,05	30	1,32	30	0,03
	nomystery	15	480	0	0,06	0	0,51	0	0,53	0	0,01
	parking	18	1663	152	0,20	152	1,17	152	1,78	152	0,03
	pegsol	19	603	0	0,09	0	1,06	0	1,10	0	0,01
	scanalyzer	18	1417	232	0,24	232	0,88	232	1,61	236	0,05
	sokoban	1	121	22	0,02	22	0,13	22	0,29	22	0,01
	tidybot	16	1224	348	0,16	348	0,84	348	2,13	350	0,08
	transport	4	1446	508	0,20	539	0,40	532	1,65	553	0,16
	woodworking	20	1325	0	0,31	0	1,11	0	1,21	0	0,01

Experiments

- We examined our methods on plans obtained by state-of-the-art satisficing planners (Fast Downward and Madagascar) for problems from the International Planning Competition (IPC 2011) domains (20 problems each).
- We used Sat4j for SAT solving and QmaxSAT for partial MaxSAT solving
- The experiments were run on a PC with Intel i7 960 cpu @3.20 Ghz and 24 GB of memory

Conclusion

- Plans obtained by satisficing planners on IPC domains often contain a lot of redundant actions
- Our new methods can remove more redundant actions than the previous approach
- Despite the NP – completeness of the problem of removing all redundant actions, all the redundant actions (even the maximum sets of redundant actions) can be eliminated very quickly.
- Thanks to the excellent performance of state-of-the-art SAT and MaxSAT solvers our SAT encoding based algorithms have very low runtimes.

24th International Conference on Automated Planning and Scheduling (ICAPS) - 2014