Eliminating All Redundant Actions from Plans Using SAT and MaxSAT

Tomáš Balyo
Lukáš Chrpa

Charles University in Prague

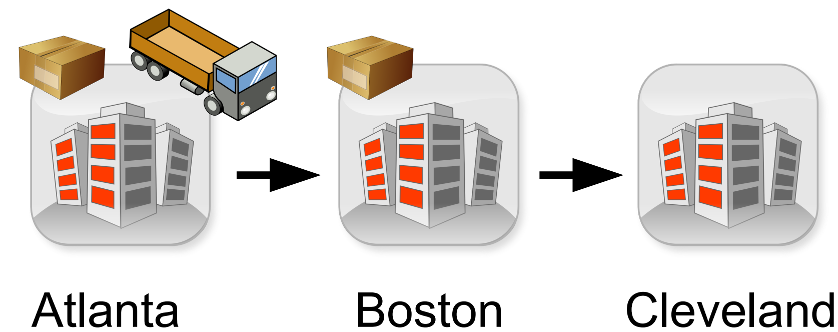
Outline

- Problem description
- Definitions – SAT, MaxSAT, SAS+
- Redundant plans
- SAT encoding of plan reduction
- Removing all redundant actions
- Removing the maximum number of redundant actions
- Experimental results on IPC 2011 domains

Problem Description

Initial State

- A package in Atlanta and Boston
- A truck in Atlanta

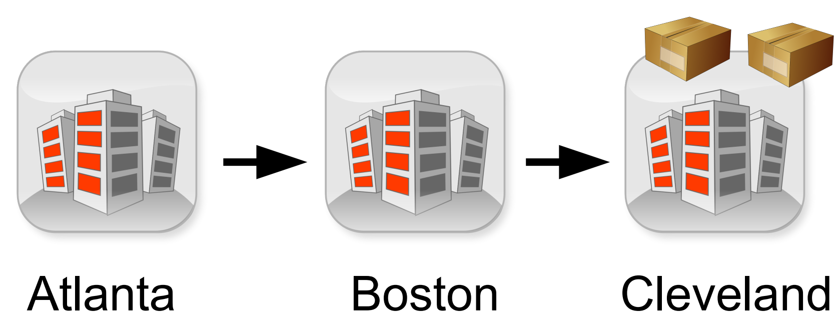


Optimal plan: `Load(P1,A), Move(A,B), Load(P2,B), Move(B,C), Unload(P1,C), Unload(P2,C)`

Shortest possible plan with 6 actions

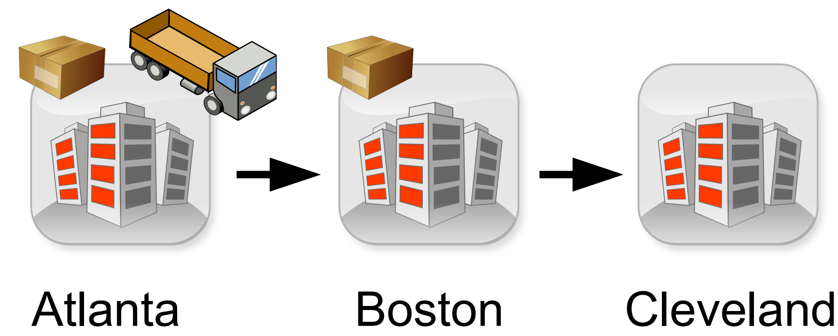
Goal State

- Both packages in Cleveland



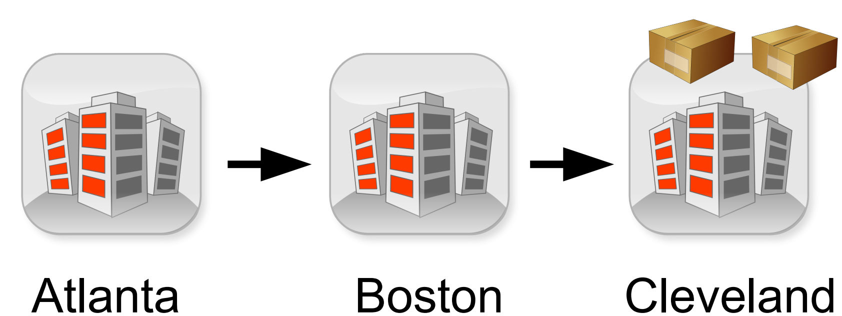
Problem Description

Initial State
- A package in Atlanta and Boston
- A truck in Atlanta



~~Optimal~~ plan: `Load(P1,A), Move(A,B), Load(P2,B), Move(B,C), Unload(P1,C), Unload(P2,C), Move(C,A)`

Goal State
- Both packages in Cleveland

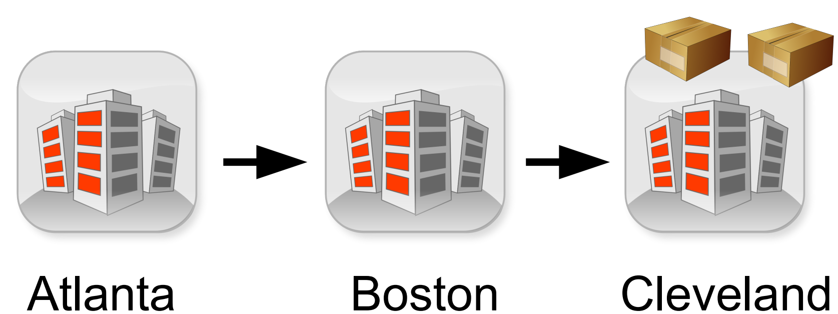


Problem Description

Initial State
- A package in Atlanta and Bo
- A truck in Atlanta

and

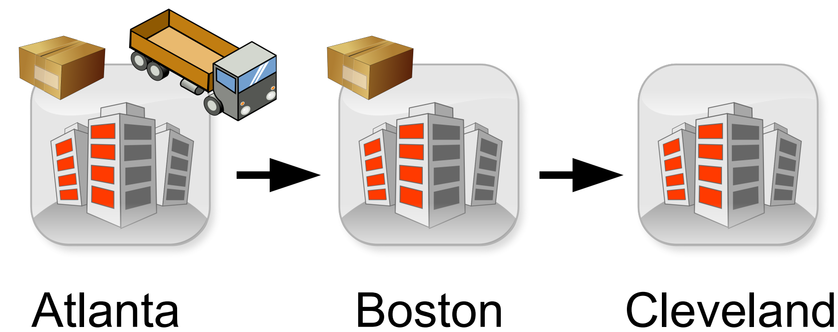
Redundant
~~Optimal~~ plan: `Load(P1,A), Move(A,B), Load(P2,B), Move(B,C), Unload(P1,C), Unload(P2,C), Move(C,A)`



Goal State
- Both packages in Cleveland

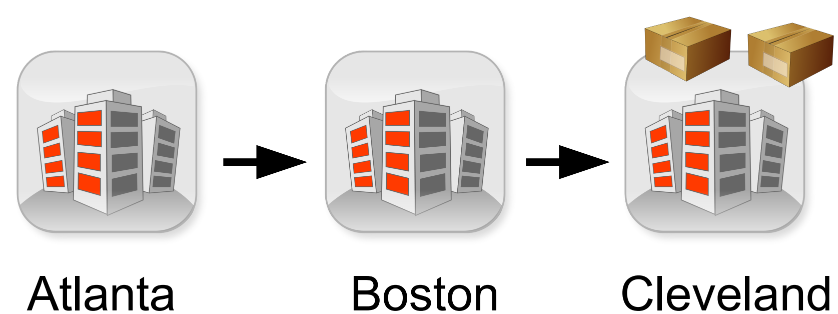
Problem Description

Initial State
- A package in Atlanta and Boston
- A truck in Atlanta



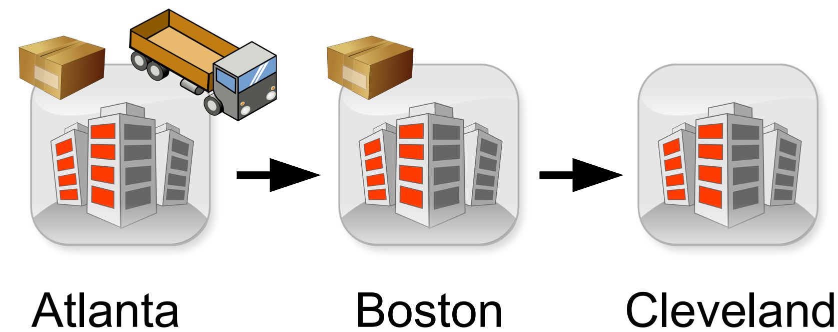
Redundant plan: `Move(A,C), Move(C,A), Load(P1,A), Move(A,B), Load(P2,B), Move(B,C), Unload(P1,C), Unload(P2,C)`

Goal State
- Both packages in Cleveland



Problem Description

Initial State
- A package in Atlanta and Boston
- A truck in Atlanta

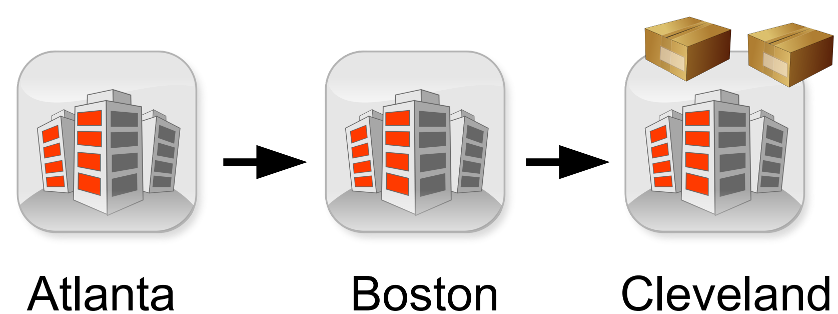


~~Redundant~~ plan:

```
Move(A,C), Move(C,B), Load(P2,B),
Move(B,A), Move(A,C), Unload(P2,C),
Move(C,B), Move(B,A), Load(P1,A),
Move(A,B), Move(B,C), Unload(P2,C)
```

12 actions, none can be removed

Goal State
- Both packages in Cleveland



Problem Description

- Our goal is to remove all redundant actions from plans in order to improve them
- After removing all redundant action, plans can be often further improved by replacing actions
  - But we will not deal with such optimization
    - There are other algorithms for that
- Plans obtained by satisficing planners often contain many redundant actions

Definitions – SAT

- A **Boolean variable** has two possible values – **true** and **false**
- A **literal** a is a Boolean variable (**positive** literal x) or its negation (**negative** literal -x)
- A **clause** is a disjunction (or) of literals
- A **CNF formula** is conjuction (and) of clauses
- A truth assignment T
  - assigns a value T(x) to each Boolean variable x
  - satisfies a positive literal x if T(x)=true and a negative literal -x if T(x)=false
  - satisfies a clause if it satisfies any of its literals
  - satisfies a CNF formula if it satisfies all of its clauses

Definitions – SAT, MaxSAT

- A CNF formula is **satisfiable** if there is a truth assignment that satisfies it
- The **Satisfiability** (**SAT**) problem is to determine whether a given formula is satisfiable (and find a truth assignmnet if yes)
- The **Maximum Satisfiability** (**MaxSAT**) problem is to find a truth assignment that satisfies as many clauses of a CNF formula as possible
- A **Partial MaxSAT** (**PMaxSAT**) formula consists of hard and soft clauses. The PmaxSAT problem is to find a truth assignment that satisfies all its hard clauses and as many of its soft clauses as possible

Definitions – SAS+

- A SAS+ planning task consists of
  - A finite set of multivalued **state variables**. Each variable has a finite domain
  - A finite set of **actions** with preconditions and effects, which are of the form x=e, where x is a state variable and e is a value from the domain of x
  - Description of the **initial state** – the initial values of all the state variables
  - A set of **goal conditions** in the form of x=e, where e is the goal value of the state variable x

Definitions – SAS+

- A **state** is a set of assignments, where each state variable has exactly one value assigned
- An action is **applicable** to a given state if all of its preconditions are compatible with the state.
- A new state S' is obtained by **applying** an action A to a state S (denoted by `S'=app(A,S)`). The values of state variables in S' are copied from S and then some of them are changed according to the effects of A
- A **plan** P is sequence of actions (P=[A1,A2,...,An]) such that the state `app(An,...app(A2,app(A1, init))...)` satisfies all the goal conditions

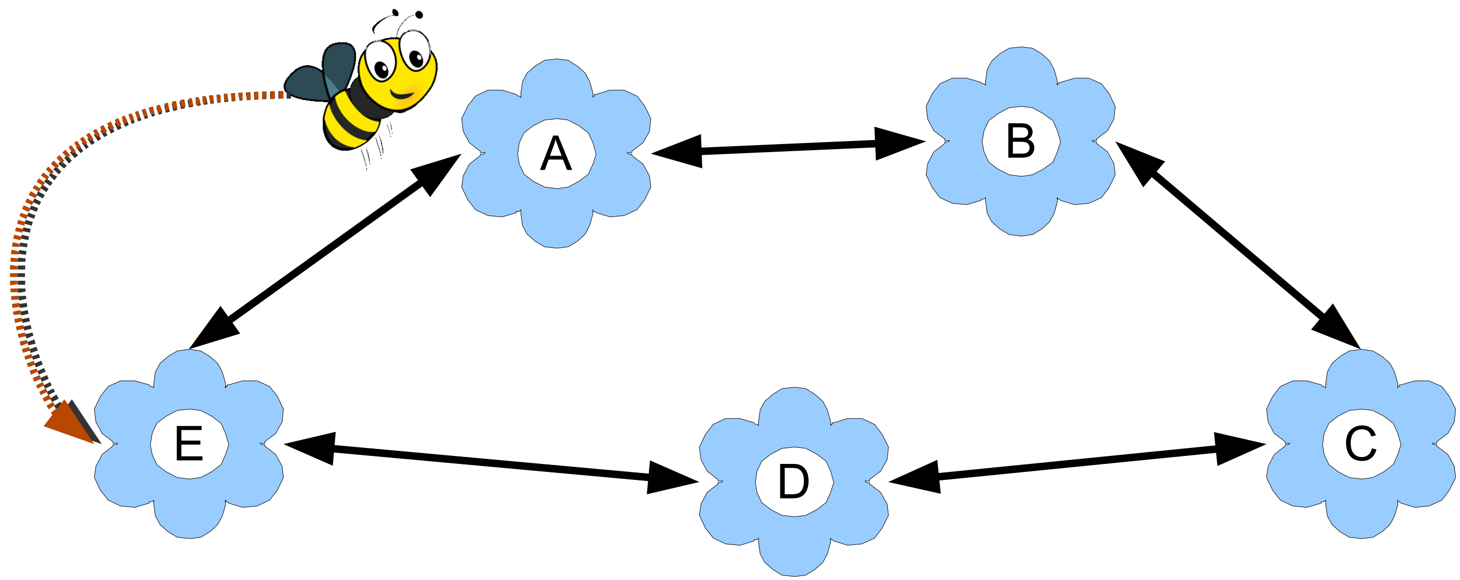
Redundant Plans

- Let P be a plan for a planning task T and let P' be a proper subsequence of P. If P' is a plan for T, then P' is called a **plan reduction** of P.
- A plan is **redundant** if it has a plan reduction
- A plan is called **perfectly justified** if it is not redundant
- Determining whether a plan is redundant is an NP complete problem (Fink, Yang 1992)

Removing Redundancy

- Prior to this work there were only incomplete heuristic algorithms
  - Removing pairs/groups of inverse actions (Chrpa, McCluskey, Osborne 2012)
  - Greedy justification (Fink, Yang 1992)
  - Action elimination (Nakhost, Müller 2010)
- We will remove all redundant actions (NP hard)
- We will remove the maximum possible number of redundant actions

Removing Redundancy



`fly(A,E), fly(E,A), fly(A,B), fly(B,C), fly(C,D), fly(D,E)`

Remove These to get a non-optimal but perferctly justified plan

Remove These to get an optimal and perferctly justified plan

- The order of removing redundant actions matters

Encoding Plan Reduction

- For a given planning task and its plan P we construct a CNF formula F such that
  - Each satisfying assignment of F represents a plan reduction of P or P itself
  - F contains a Boolean variable for each action in P which indicates, whether the action is present in the plan reduction
- By adding the clause $\left(\neg a_1 \vee \neg a_2 \vee \ldots \vee \neg a_n\right)$ to F we obtain a formula that is satisfiable if and only if P is a redundant plan

SAT-based Redundancy Elimination

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

SAT-based Redundancy Elimination
Incremental Version

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

Removing The Maximum Number of Redundant Actions

- We will use Partial MaxSAT solving
  - The hard clauses are the plan reducion encoding
  - The soft clauses are unit clauses

$$(\neg a_1), (\neg a_2), \ldots (\neg a_n)$$

- The PmaxSAT solver will satisfy all the hard clauses and as many soft clauses as possible, i.e., remove as many actions as possible

MaximumRedundancyEliminaion (Π, P)

$F :=$ encodeMaximumRedundancy(Π, P)

$\phi :=$ partialMaxSatSolver(F)

return P_ϕ

Experiments

- We used 3 satisficing planners
  - Metric FF
  - Fast Downward
  - Madagascar
- 10 minute time limit to find plans for each problem of the 2011 IPC
- Plan reduction methods
  - Action elimination
  - SAT-based reduction
  - PmaxSAT-based reduction

	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

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 approaches
- 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