On Different Strategies for Eliminating Redundant Actions from Plans

Tomáš Balyo, Lukáš Chrpa, Asma Kilani

Karlsruhe Institute of Technology

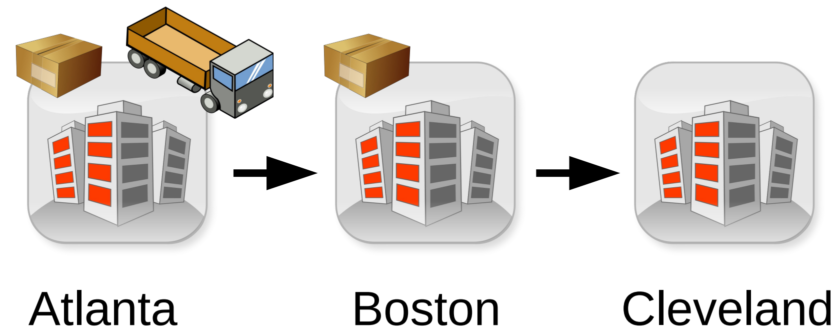
University of Huddersfield

Outline

- Problem description
- Definitions – SAT, MaxSAT, SAS+
- Redundant plans
- Heuristic approaches
- SAT encoding of plan reduction
- Removing the largest and most costly sets of redundant actions
- Experimental results on IPC 2011 domains
- Conclusion

Problem Description

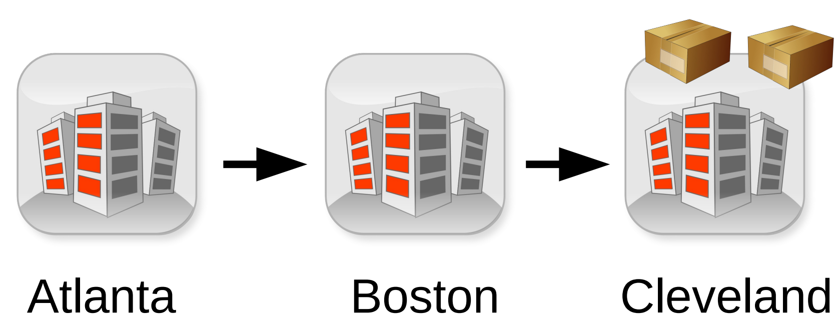
Initial State
- A package in Atlanta and Boston
- A truck in Atlanta



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

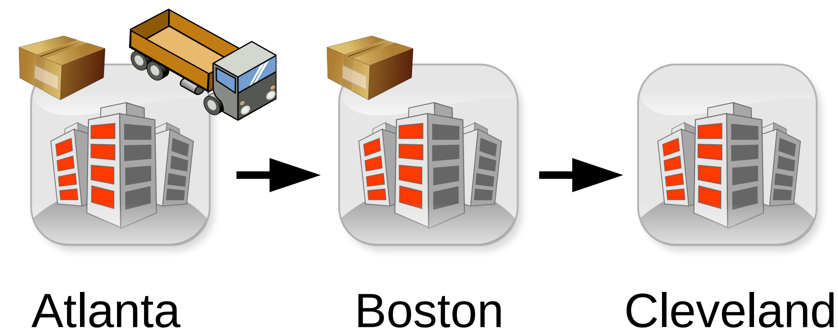
Goal State
- Both packages in Cleveland



Problem Description

Initial State

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

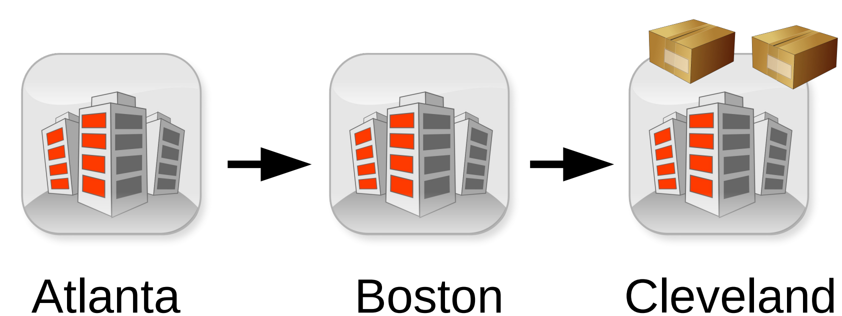


~~Optimal~~ plan:

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

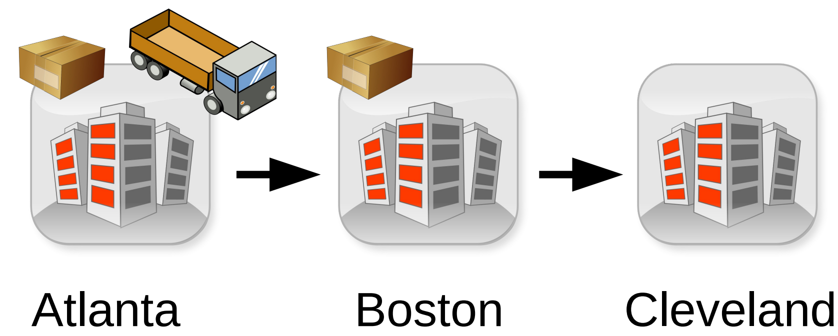
Goal State

- Both packages in Cleveland



Problem Description

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

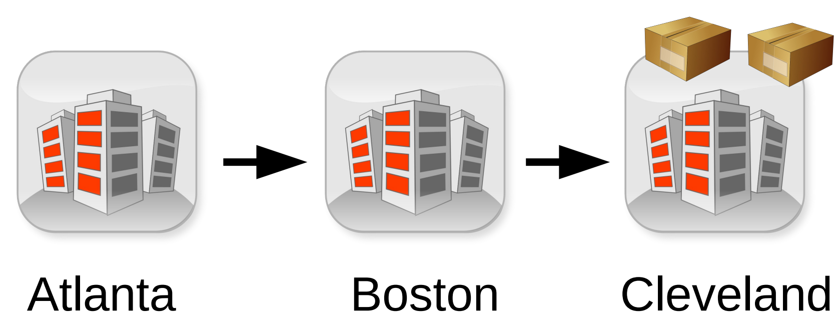


Redundant

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

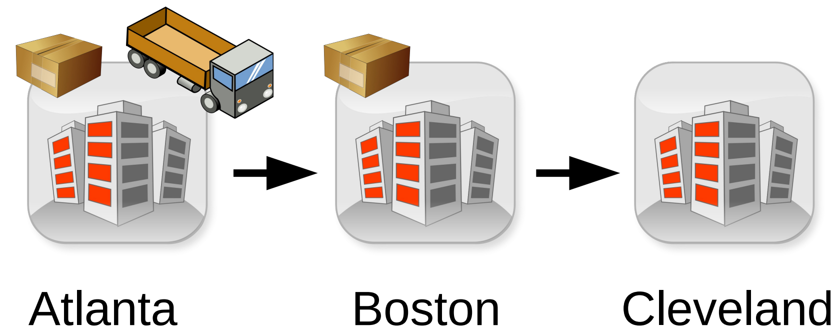
Why is this “move“ in the plan?

Goal State
- Both packages in Cleveland



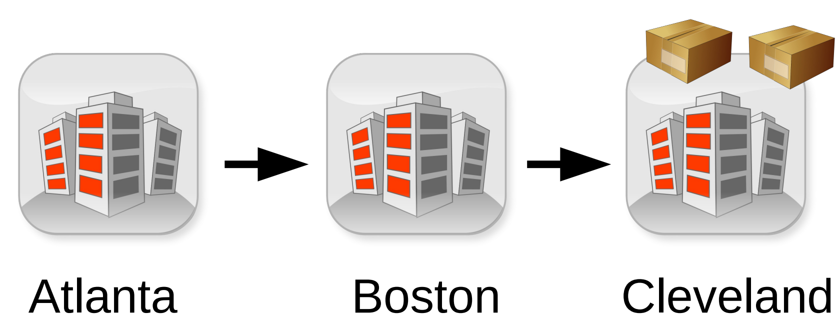
Problem Description

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



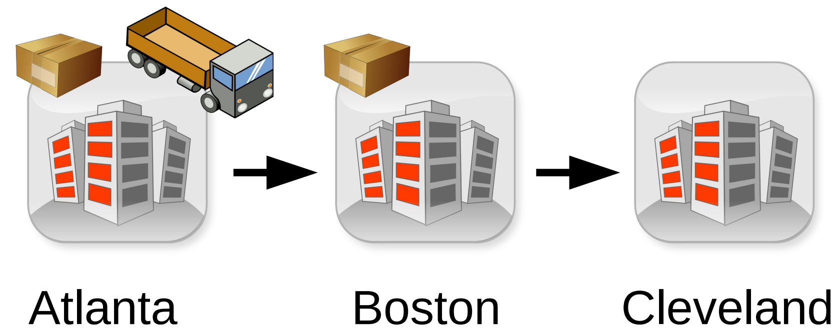
Redundant plan: `Move(A,C), Move(C,A), Load(P1,A), Move(A,B), Load(P2,B), Move(B,C), Unload(P1,C), Unload(P2,C)`

Goal State
- Both packages in Cleveland



Problem Description

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

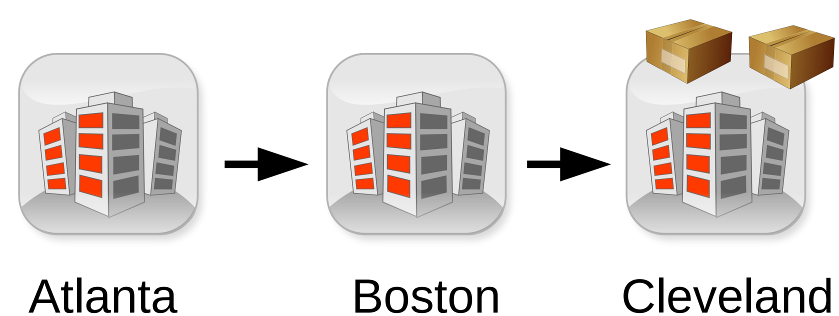


~~Redundant~~ plan:

```
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 actions, plans can be often further improved by replacing or reordeing (and further removing) 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)
- 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
- A **Weighted Partial MaxSAT** (**WPMaxSAT**) is like PMaxSAT, but the soft clauses have weights and the goal is to maximize the weight of the satisfied soft clauses

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
- Actions have costs, the total cost of a plan is the sum of the costs of its actions

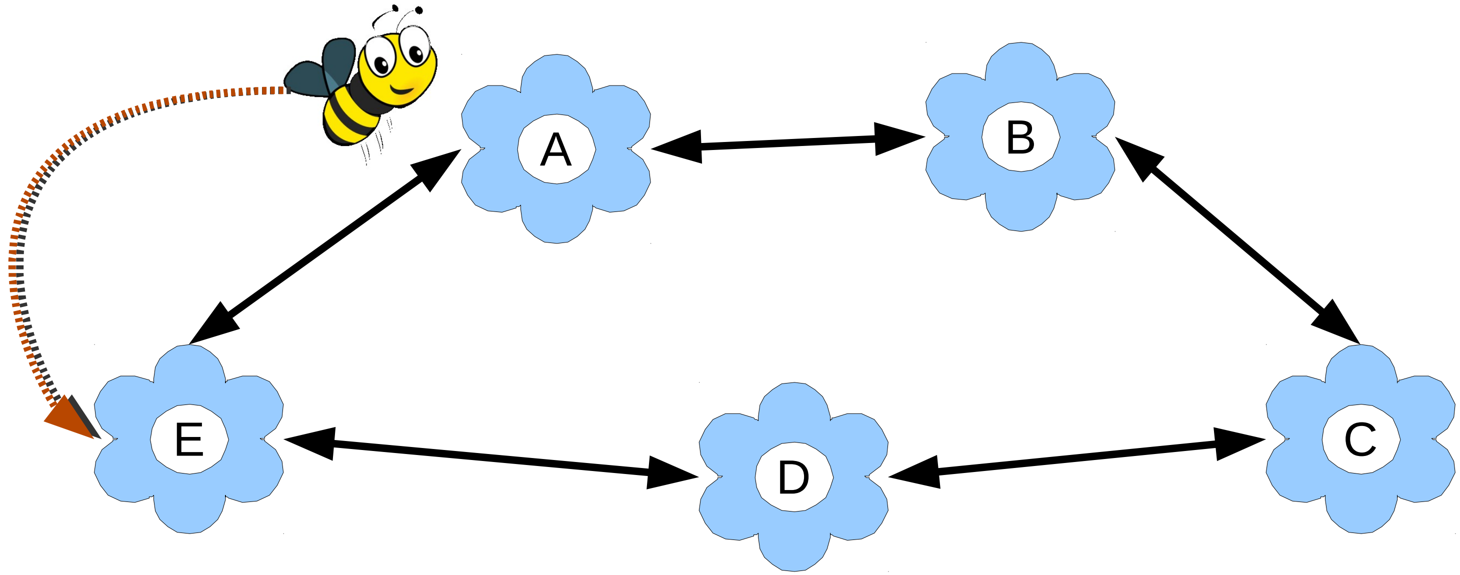
Redundant Plans

- 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 introduce our own heuristic algorithm
- We will then show how remove the set of redundant actions with a maximum possible total cost (NP-hard)

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

[Greedy] Action Elimination

evaluateRemove (Π, P, k)

```
s := s_I
for i := 1 to k − 1 do
    s := apply(P[i],s)
cost := C(P[k])
for i := k + 1 to |P| do
    if applicable(P[i], s) then
        s := apply(P[i], s)
    else
        cost := cost + C(P[i])
if goalSatisfied(Π, s) then
    return cost
else
    return −1
```

greedyActionElimination (Π, P)

```
repeat
    bestCost := 0
    bestIndex := 0
    for i := 1 to |P| do
        cost := evaluateRemove(Π, P, i)
        if cost ≥ bestCost then
            bestCost := cost
            bestIndex := i
    if bestIndex ≠ 0 then
        P := remove(P, bestIndex)
until bestIndex = 0
return P
```

remove (P, k)

```
s := s_I
P′ := [ ]     // empty plan
for i := 1 to k − 1 do
    s := apply(P[i],s)
    P′ := append(P′,P[i])
for i := k + 1 to |P| do
    if applicable(P[i], s) then
        s := apply(P[i], s)
        P′ := append(P′, P[i])
return P′
```

- Can we remove the k-th action and all that depend on it? How much would we gain?
- Remove the set with the maximal gain

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 a_j for each action in P which indicates the presence of the *j*-th action 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

Encoding – basic ideas

- We need to ensure that a given condition holds at a given time
  - Goal conditions in the end
  - Action preconditions when the action is applied
- Two ways to ensure a condition C at time T
  - Either C is an initial condition and there are no opposing actions in the plan reduction before T
  - Or there is a supporting action in the reduction at time $T' < T$ for C and there are no opposing actions between T' and T

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

Removing The Set of Redundant Actions with Maximum Weight

- We will use Weighted Partial MaxSAT solving
  - The hard clauses are the plan reducion encoding
  - The soft clauses are unit clauses, weight = act. cost

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

- The WPmaxSAT solver will satisfy all the hard clauses and maximize the weight of the satisfied soft clauses, i.e., remove the most costly set of redundant actions.

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
  - Inverse Action Elimination
  - Action Elimination and Greedy Action Elimination
  - PMaxSAT and WPMaxSAT reduction

Experimental Results

Domain		Found Plan Nr.	Found Plan Cost	IAE Δ	IAE T[s]	AE Δ	AE T[s]	Greedy AE Δ	Greedy AE T[s]	MLR Δ	MLR T[s]	MR Δ	MR T[s]
Fast Downward	barman	20	7763	436	0,98	753	0,51	780	1,08	926	0,43	926	10,85
	elevators	20	28127	1068	1,51	1218	0,79	1218	1,20	1218	0,19	1218	1,99
	floortile	5	572	66	0,00	66	0,04	66	0,08	66	0,00	66	0,01
	nomystery	13	451	0	4,25	0	0,04	0	0,04	0	0,01	0	0,04
	parking	20	1494	4	0,06	4	0,09	4	0,10	4	0,04	4	0,21
	pegsol	20	307	0	0,00	0	0,06	0	0,06	0	0,02	0	0,30
	scanalyzer	20	1785	0	0,01	78	0,06	78	0,08	78	0,04	78	0,49
	sokoban	17	1239	0	6,48	58	0,53	58	0,75	102	1,92	102	250,27
	transport	17	74960	4194	1,11	5259	0,56	5260	1,02	5260	0,19	5260	1,92
Madagascar	barman	8	3360	296	0,97	591	0,25	598	0,52	606	0,28	606	6,33
	elevators	20	117641	7014	6,77	24096	1,21	24728	10,44	28865	1,90	28933	37,34
	floortile	20	4438	96	0,09	96	0,31	96	0,37	96	0,04	96	0,24
	nomystery	15	480	0	2,63	0	0,04	0	0,04	0	0,01	0	0,02
	parking	18	1663	152	0,17	152	0,12	152	0,40	152	0,04	152	0,36
	pegsol	19	280	0	0,00	0	0,05	0	0,06	0	0,01	0	0,26
	scanalyzer	18	1875	0	0,05	232	0,19	236	0,47	236	0,04	236	0,31
	sokoban	1	33	0	0,01	0	0,02	0	0,04	0	0,01	0	0,19
	transport	4	20496	4222	0,23	6928	0,20	7507	0,56	7736	0,16	7736	9,56

Conclusion

- Plans obtained by satisficing planners on IPC domains often contain a lot of redundant actions
- Our new methods can improve the cost of a plan more than the previous approaches (restricted to elimination)
- Despite the NP – completeness of the problem of removing a maximum set of redundant actions, our methods are very fast on IPC problems (thanks to the excellent performance of state-of-the-art MaxSAT solvers)
- Future work
  - Allow the reordering of actions before redundancy elimination to eliminate more actions
  - Find ways of detecting the cases when (G)AE achieves optimal elimination, i.e., polynomial methods are sufficient