Modelling and Solving Problems Using SAT Techniques

Tomáš Balyo

Modelling and Solving Problems Using SAT Techniques

Planning

Tomáš Balyo

What is Planning

State Variables and their domains:

- Truck location **T**, dom(**T**)={A, B, C}
- Package locations **P** and **Q**

dom(**P**) = dom(**Q**) = {A, B, C, Tr}

Initial State: **T**=A, **P**=A, **Q**=B
Goal State: **P**=C, **Q**=C

Actions:

- `move(x,y)`: {T=x} ~> {T=y}
- `loadP(x)`: {T=x, P=x} ~> {P=Tr}
- `loadQ(x)`: {T=x, Q=x} ~> {Q=Tr}
- `dropP(x)`: {T=x, P=Tr} ~> {P=x}
- `dropQ(x)`: {T=x, Q=Tr} ~> {Q=x}

Where `x`, `y` are A, B, and C

Plan: `loadP(A), move(A,B), loadQ(B), move(B,C), dropP(C), dropQ(C)`

Planning as SATisfiability

- Construct a formula $\mathbf{F_k}$ such that it is satisfiable (if and) only if there is a plan of at most **k** steps
- Solve $\mathbf{F_1}$, $\mathbf{F_2}$, … using a SAT solver until you reach a satisfiable formula $\mathbf{F_n}$
- Extract a plan from the satisfying assignment of $\mathbf{F_n}$
- **n** is called the makespan of the plan

- ***What actions can go inside a step together?***
  - If more action could be in a step then we would need fewer steps to find a plan

What actions can go inside a step together?

1. foreach step semantics

- The preconditions of all actions in a step must already hold in the beginning of the step
- The effects of all actions must hold at the end of this step
- The actions in a step do not interfere – they cannot destroy each others preconditions by their effects => can be ordered arbitrarily
- The actions in a step can be turned into a valid subplan sequence

- Plan: `{loadP(A)}` ♦ `{move(A, B)}` ♦ `{loadQ(B)}` ♦ `{move(B, C)}` ♦ `{dropP(C), dropQ(C)}`
  – **5 steps**

What actions can go inside a step together?

2. exist step semantics

- The preconditions of all actions in a step must already hold in the beginning of the step
- The effects of all actions must hold at the end of this step
- ~~The actions in a step do not interfere – they cannot destroy each others preconditions by their effects => can be ordered arbitrarily~~
- The actions in a step can be turned into a valid subplan sequence

- Plan: `{loadP(A), move(A, B)}` ♦ `{loadQ(B), move(B, C)}` ♦ `{dropP(C), dropQ(C)}`
  – **3 steps**

What actions can go inside a step together?

3. relaxed exist step semantics

- ~~The preconditions of all actions in a step must already hold in the beginning of the step~~
- The effects of all actions must hold at the end of this step
- ~~The actions in a step do not interfere – they cannot destroy each others preconditions by their effects => can be ordered arbitrarily~~
- The actions in a step can be turned into a valid subplan sequence

- Plan: {`loadP(A), move(A, B), loadQ(B)`}♦ {`move(B, C), dropP(C), dropQ(C)`}
  – **2 steps**

What actions can go inside a step together?

4. relaxed relaxed exist step semantics

New!

- ~~The preconditions of all actions in a step must already hold in the beginning of the step~~
- ~~The effects of all actions must hold at the end of this step~~
- ~~The actions in a step do not interfere – they cannot destroy each others preconditions by their effects => can be ordered arbitrarily~~
- The actions in a step can be turned into a valid subplan sequence

- Plan: {`loadP(A), move(A, B), loadQ(B), move(B, C), dropP(C), dropQ(C)`}
  – **1 step**

Implemented SAT Encodings

- We implemented 3 foreach step semantics encodings:
  - Direct (classic)
  - SASE (transition based)
  - Reinforced (Direct + SASE) * New!
- A Relaxed Relaxed Exist Step semantics encoding ** New!
- A Selective encoding which automatically selects * or ** for a given planning problem instance New!

- The selective encoding can outperform the state-of-the-art exist step encoding (of Rintanen 2006).

Table 3.3: The number of problems in each domain that the encodings solved within the time limit (30 minutes for SAT solving).

Domain	Dir	SASE	Reinf	$R^2\exists$	Sel	R∀	R∃
barman	4	4	4	8	**9**	8	4
elevators	**20**	**20**	**20**	**20**	**20**	**20**	**20**
floortile	16	11	18	18	18	16	**20**
nomystery	**20**	10	**20**	6	**20**	**20**	**20**
openstacks	0	0	0	15	**20**	0	0
parcprinter	**20**	**20**	**20**	**20**	**20**	**20**	**20**
parking	0	0	0	0	0	0	0
pegsol	10	6	10	**19**	**19**	11	12
scanalyzer	14	12	15	9	15	17	**18**
sokoban	2	2	2	2	2	**6**	**6**
tidybot	2	2	2	2	2	13	**15**
transport	16	17	18	13	**19**	18	18
visitall	12	9	10	**20**	**20**	11	11
woodworking	**20**	**20**	**20**	**20**	**20**	**20**	**20**
Total	156	133	159	172	**204**	180	184

Basic ideas of the relaxed relaxed exist step SAT encoding

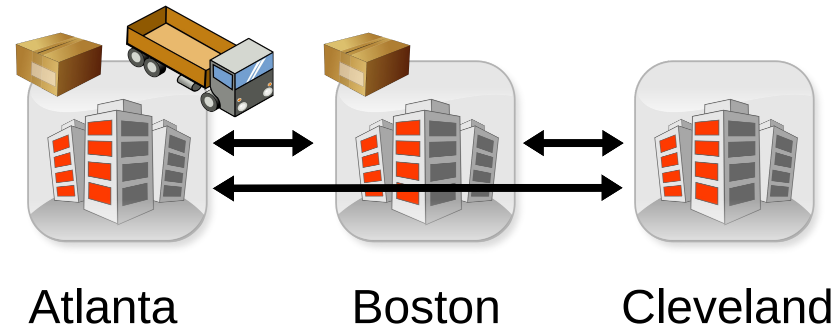
- The SAT encoding only approximates the semantics, i.e., the satisfiability of the constructed formula $\mathbf{F}_k$ implies the existence of a **k**-step plan (not vice versa)
- The actions are ranked – the encoding allows only lower ranking actions before higher ranking ones in a step (the reason why the encoding only approximates the semantics)
- The ranking can be an arbitrary injective function, some rankings are better than others for some problems
  - A perfect ranking could be created if we knew the plan in advance

Part II – Removing Redundant Actions

(From plans obtained by any planner)

Problem Description

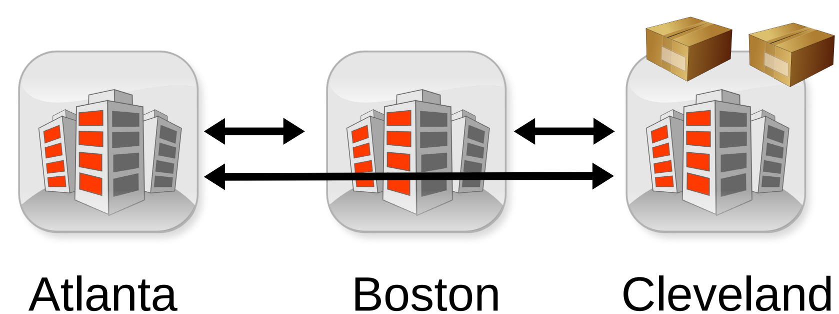
Initial State
- A package in Atlanta and Boston
- A truck in Atlanta



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

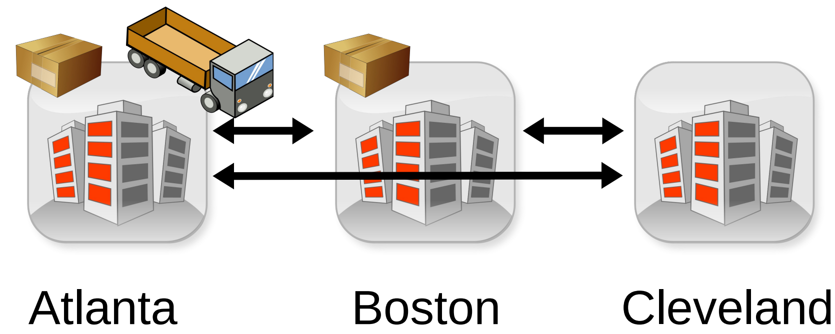
Goal State
- Both packages in Cleveland



Problem Description

Initial State

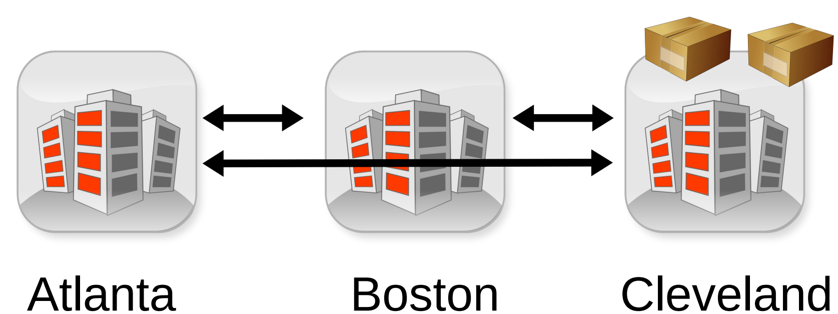
- A package in Atlanta and Boston
- A truck in Atlanta



~~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

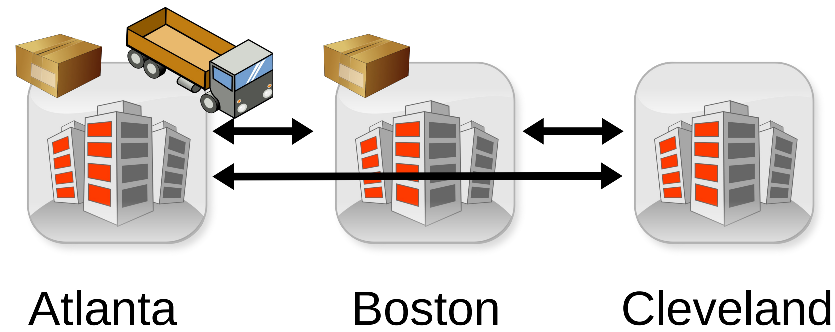
- Both packages in Cleveland



Problem Description

Initial State

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



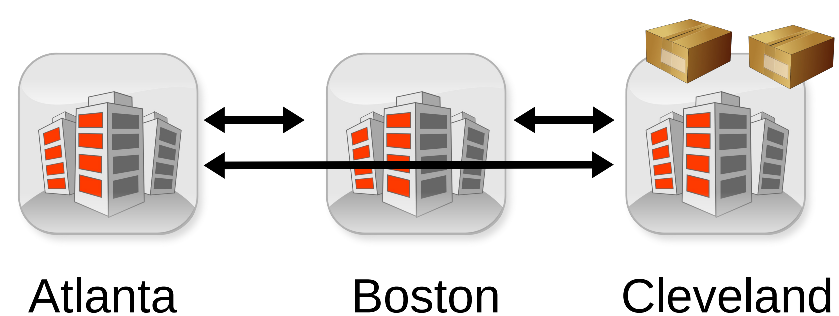
Redundant

~~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

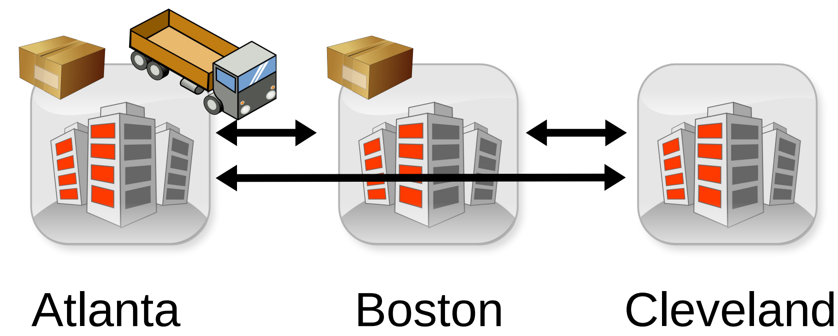
- Both packages in Cleveland



Problem Description

Initial State

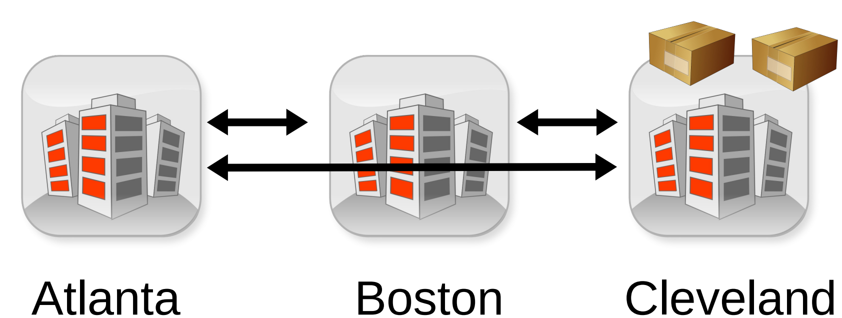
- A package in Atlanta and Boston
- A truck in Atlanta



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

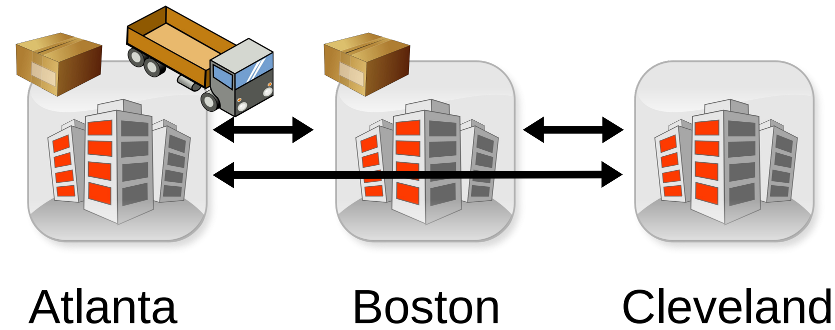
- Both packages in Cleveland



Problem Description

Initial State

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



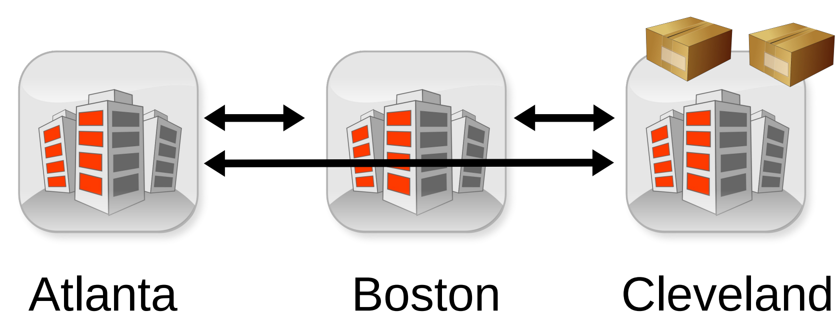
~~Redundant~~ plan:

```
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(P1,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, future work
- Plans obtained by satisficing planners often contain many redundant actions

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 assignment 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

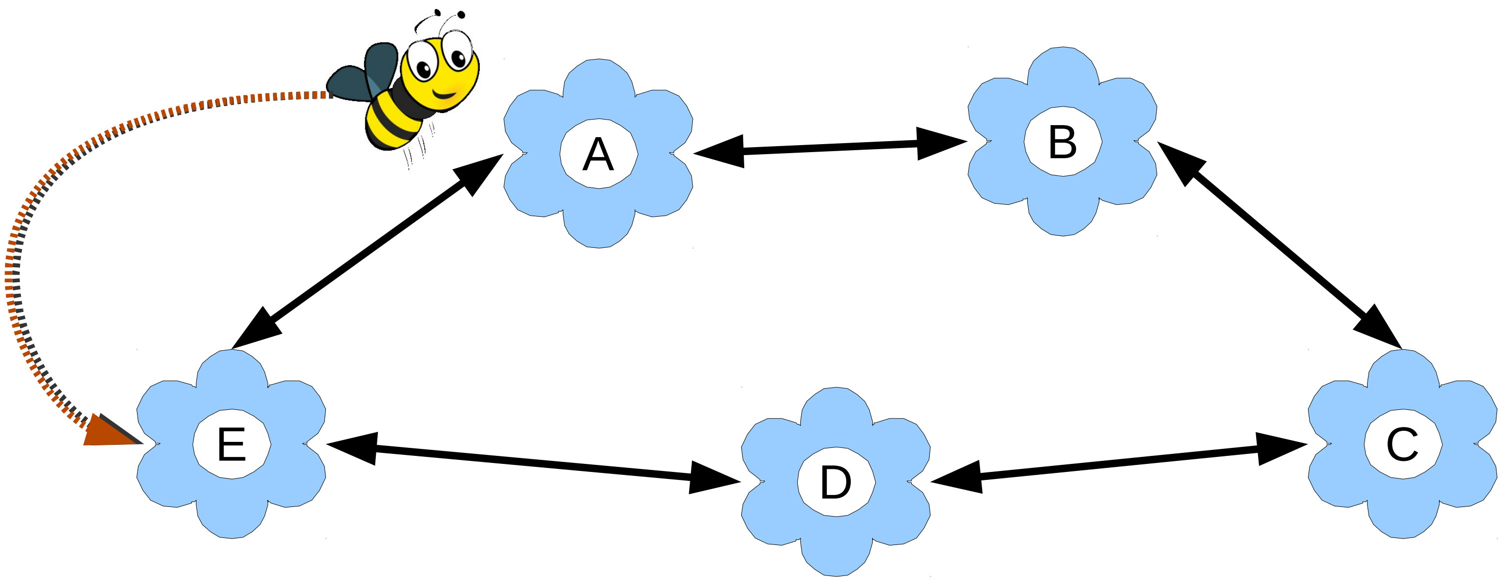
Redundant Plans

- 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
- The actions not present in a plan reduction are redundant actions
- 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 also non-redundant plan

Remove These to get an optimal and non-redundant plan

- The order of removing redundant actions matters

[Greedy] Action Elimination

- Polynomial heuristic algorithm for removing redundant actions

```
FOR i := 1 to |P| DO
  P' := remove(P, P[i])
  remove-actions-with-unsat-preconditions(P')
  IF (P' is a valid plan) THEN P:=P'
DONE
---------------------------------------------
Repeat
  S:={}
  FOR i := 1 to |P| DO
    P' := remove(P, P[i])
    remove-actions-with-unsat-preconditions(P')
    IF (P' is a valid plan) THEN insert(S,P')
  DONE
  P:= best-of(S)
UNTIL S:={}
```

New!

Encoding Plan Reduction

- For a given planning task and its plan P we construct a CNF formula F such that

New!

  - 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 $(\neg a_1 \vee \neg a_2 \vee \ldots \vee \neg a_n)$ 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 reduction 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

New!

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.

New!

Experiments

- We used 2 satisficing planners
  - 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

- In the thesis we have introduced new methods for finding plans and improving plans using SAT and MaxSAT solvers
- A combination of our encodings outperforms the encodings used in state-of-the-art SAT-based planners
- Our plan improvement methods can improve the cost and length of plans more than the previous approaches (restricted to redundancy 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)