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

Goal State
• Both packages in Cleveland

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

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

Goal State
• Both packages in Cleveland

Optimal plan: \( \text{Load}(P1,A), \text{Move}(A,B), \text{Load}(P2,B), \text{Move}(B,C), \text{Unload}(P1,C), \text{Unload}(P2,C), \text{Move}(C,A) \)
Problem Description

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

Goal State
- Both packages in Cleveland

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

Redundant
Why is that last "move" in the plan?
Problem Description

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

Goal State
- Both packages in Cleveland

Redundant plan:
\[
Move(A, C), \ Move(C, A), \ Load(P1, A), \ Move(A, B), \ Load(P2, B), \ Move(B, C), \ Unload(P1, C), \ Unload(P2, C)
\]
Problem Description

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

Goal State
- Both packages in Cleveland

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
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 conjunction (and) of clauses
- A truth assignment $T$
  - assigns a value $T(x)$ to each Boolean variable $x$
  - satisfies a positive literal $x$ if $T(x) = \text{true}$ and a negative literal $-x$ if $T(x) = \text{false}$
  - 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 assignment 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 $S' = \text{app}(A, S)$). 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 ($P = [A_1, A_2, ..., A_n]$) such that the state $\text{app}(A_n, \ldots \text{app}(A_2, \text{app}(A_1, \text{init})))$ 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

\[
\text{fly}(A, E), \, \text{fly}(E, A), \, \text{fly}(A, B), \, \text{fly}(B, C), \, \text{fly}(C, D), \, \text{fly}(D, E)
\]

- Remove These to get a non-optimal but perfectly justified plan
- Remove These to get an optimal and perfectly 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 \((\neg a_1 \lor \neg a_2 \lor ... \lor \neg a_n)\) to F we obtain a formula that is satisfiable if and only if P is a redundant plan
SAT-based Redundancy Elimination

\textbf{RedundancyElimination}(\Pi, P)

\begin{align*}
&I_1 \quad F_{\Pi, P} := \text{encodeRedundancy}(\Pi, P) \\
&I_2 \quad \textbf{while} \ \text{isSatisfiable}(F_{\Pi, P}) \ \textbf{do} \\
&I_3 \quad \phi := \text{getSatAssignment}(F_{\Pi, P}) \\
&I_4 \quad P := P_{\phi} \\
&I_5 \quad F_{\Pi, P} := \text{encodeRedundancy}(\Pi, P) \\
&I_6 \quad \textbf{return} \ P
\end{align*}
SAT-based Redundancy Elimination
Incremental Version

\begin{verbatim}
IncrementalRedundancyElimination (Π, P)
solver = new SatSolver
solver.addClauses(encodeRedundancy(Π, P))
while solver.isSatisfiable() do
    φ := solver.getSatAssignment()
    C := \bigvee \{¬a_i | a_i \in P_φ \}
    solver.addClause(C)
    foreach a_i \in P do if φ(a_i) = False then
        solver.addClause(\{¬a_i \})
    P := P_φ
return P
\end{verbatim}
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

\[
\text{MaximumRedundancyElimination}(\Pi, P)
\]

MR1 \quad F := \text{encodeMaximumRedundancy}(\Pi, P)

MR2 \quad \phi := \text{partialMaxSatSolver}(F)

MR3 \quad \text{return } P_\phi
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
<table>
<thead>
<tr>
<th>Domain</th>
<th>#Plans</th>
<th>Length</th>
<th>$\Delta_{AE}$</th>
<th>$T_{AE}$</th>
<th>$\Delta_{AE+S}$</th>
<th>$T_{AE+S}$</th>
<th>$\Delta_{SAT}$</th>
<th>$T_{SAT}$</th>
<th>$\Delta_{MAX}$</th>
<th>$T_{MAX}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>barman</td>
<td>20</td>
<td>3749</td>
<td>528</td>
<td>0.52</td>
<td>582</td>
<td>3.44</td>
<td>596</td>
<td>7.18</td>
<td>629</td>
<td>0.44</td>
</tr>
<tr>
<td>elevators</td>
<td>20</td>
<td>4625</td>
<td>94</td>
<td>0.84</td>
<td>94</td>
<td>2.41</td>
<td>94</td>
<td>3.45</td>
<td>94</td>
<td>0.19</td>
</tr>
<tr>
<td>floortile</td>
<td>5</td>
<td>234</td>
<td>22</td>
<td>0.06</td>
<td>22</td>
<td>0.20</td>
<td>22</td>
<td>0.27</td>
<td>22</td>
<td>0.00</td>
</tr>
<tr>
<td>nomystery</td>
<td>13</td>
<td>451</td>
<td>0</td>
<td>0.05</td>
<td>0</td>
<td>0.47</td>
<td>0</td>
<td>0.48</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>parking</td>
<td>20</td>
<td>1494</td>
<td>4</td>
<td>0.17</td>
<td>4</td>
<td>1.21</td>
<td>4</td>
<td>1.26</td>
<td>4</td>
<td>0.03</td>
</tr>
<tr>
<td>pegsol</td>
<td>20</td>
<td>644</td>
<td>0</td>
<td>0.11</td>
<td>0</td>
<td>1.11</td>
<td>0</td>
<td>1.18</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>scanalyzer</td>
<td>20</td>
<td>823</td>
<td>26</td>
<td>0.10</td>
<td>26</td>
<td>1.16</td>
<td>26</td>
<td>1.33</td>
<td>26</td>
<td>0.03</td>
</tr>
<tr>
<td>sokoban</td>
<td>17</td>
<td>5094</td>
<td>244</td>
<td>0.62</td>
<td>458</td>
<td>5.25</td>
<td>458</td>
<td>8.39</td>
<td>460</td>
<td>1.84</td>
</tr>
<tr>
<td>tidybot</td>
<td>16</td>
<td>1046</td>
<td>64</td>
<td>0.14</td>
<td>64</td>
<td>0.91</td>
<td>64</td>
<td>1.28</td>
<td>64</td>
<td>0.03</td>
</tr>
<tr>
<td>transport</td>
<td>17</td>
<td>4059</td>
<td>289</td>
<td>0.65</td>
<td>289</td>
<td>1.64</td>
<td>289</td>
<td>2.93</td>
<td>290</td>
<td>0.20</td>
</tr>
<tr>
<td>visitall</td>
<td>20</td>
<td>28776</td>
<td>122</td>
<td>3.66</td>
<td>122</td>
<td>9.47</td>
<td>122</td>
<td>12.89</td>
<td>122</td>
<td>7.77</td>
</tr>
<tr>
<td>woodworking</td>
<td>20</td>
<td>1605</td>
<td>27</td>
<td>0.41</td>
<td>27</td>
<td>1.16</td>
<td>27</td>
<td>1.33</td>
<td>30</td>
<td>0.03</td>
</tr>
<tr>
<td>barman</td>
<td>8</td>
<td>1785</td>
<td>303</td>
<td>0.25</td>
<td>303</td>
<td>1.59</td>
<td>303</td>
<td>3.53</td>
<td>318</td>
<td>0.30</td>
</tr>
<tr>
<td>elevators</td>
<td>20</td>
<td>11122</td>
<td>2848</td>
<td>1.46</td>
<td>3017</td>
<td>4.13</td>
<td>3021</td>
<td>17.62</td>
<td>3138</td>
<td>2.03</td>
</tr>
<tr>
<td>floortile</td>
<td>20</td>
<td>1722</td>
<td>30</td>
<td>0.39</td>
<td>30</td>
<td>1.05</td>
<td>30</td>
<td>1.32</td>
<td>30</td>
<td>0.03</td>
</tr>
<tr>
<td>nomystery</td>
<td>15</td>
<td>480</td>
<td>0</td>
<td>0.06</td>
<td>0</td>
<td>0.51</td>
<td>0</td>
<td>0.53</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>parking</td>
<td>18</td>
<td>1663</td>
<td>152</td>
<td>0.20</td>
<td>152</td>
<td>1.17</td>
<td>152</td>
<td>1.74</td>
<td>152</td>
<td>0.03</td>
</tr>
<tr>
<td>pegsol</td>
<td>19</td>
<td>603</td>
<td>0</td>
<td>0.09</td>
<td>0</td>
<td>1.06</td>
<td>0</td>
<td>1.10</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>scanalyzer</td>
<td>18</td>
<td>1417</td>
<td>232</td>
<td>0.24</td>
<td>232</td>
<td>0.88</td>
<td>232</td>
<td>1.61</td>
<td>236</td>
<td>0.05</td>
</tr>
<tr>
<td>sokoban</td>
<td>1</td>
<td>121</td>
<td>22</td>
<td>0.02</td>
<td>22</td>
<td>0.13</td>
<td>22</td>
<td>0.29</td>
<td>22</td>
<td>0.01</td>
</tr>
<tr>
<td>tidybot</td>
<td>16</td>
<td>1224</td>
<td>348</td>
<td>0.16</td>
<td>348</td>
<td>0.84</td>
<td>348</td>
<td>2.13</td>
<td>350</td>
<td>0.08</td>
</tr>
<tr>
<td>transport</td>
<td>4</td>
<td>1446</td>
<td>508</td>
<td>0.20</td>
<td>539</td>
<td>0.40</td>
<td>532</td>
<td>1.65</td>
<td>553</td>
<td>0.16</td>
</tr>
<tr>
<td>woodworking</td>
<td>20</td>
<td>1325</td>
<td>0</td>
<td>0.31</td>
<td>0</td>
<td>1.11</td>
<td>0</td>
<td>1.21</td>
<td>0</td>
<td>0.01</td>
</tr>
</tbody>
</table>
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.