Reducción de la Planificación Conformante a SAT mediante Compilación a d–DNNF

Héctor Palacios
UPF

Héctor Geffner
ICREA/UPF
Planning

- Agent performs \textit{actions} to achieve a \textit{goal}
- Many flavors: uncertainty, time, resources, etc
- Last decade: shift from \textit{theoretical} to \textit{empirical} based. significant improvement
Planning

- Agent performs **actions** to achieve a **goal**

- Many flavors: uncertainty, time, resources, etc

- Last decade: shift from **theoretical** to **empirical** based. significant improvement

- *Classical Planning*: simplest flavor

  From **a** initial state, reach a goal by doing a plan (**sequence** of actions)

  Example: Robot navigation: starts from a position, has a map
Planning

- Agent performs **actions** to achieve a **goal**
- Many flavors: uncertainty, time, resources, etc
- Last decade: shift from **theoretical** to **empirical** based. significant improvement

- **Classical Planning**: simplest flavor
  - From a initial state, reach a goal by doing a plan (**sequence** of actions)
  - Example: Robot navigation: starts from a position, has a map

- **Conformant Planning**: slight uncertainty
  - **Many possible initial** states: one plan working for **every** initial state
  - Example: a blind Robot has a map, but doesn’t know its initial position
Motivation

- Classical Planning as SAT
  - Obtain a formula from a problem, call a solver
  - Very successful!
Motivation

- Classical Planning as SAT
  - Obtain a formula from a problem, call a solver
  - Very successful!

- Conformant Planning is NP-hard: can’t be mapped to one SAT
  - We want a formula to feed a SAT solver
  - Obtaining can be expensive
Motivation

• Classical Planning as SAT
  – Obtain a formula from a problem, call a solver
  – Very successful!

• Conformant Planning is NP-hard: can’t be mapped to one SAT
  – We want a formula to feed a SAT solver
  – Obtaining can be expensive

• We present a optimal conformant planner: obtain a formula, SAT

• The planner just need two off-the-shelf components:
  a knowledge compiler and a SAT solver

  No specific search algorithm!
Outline

- Classical Planning as SAT
- Conformant Planning as SAT
- A propositional formula for solving Conformant Planning as SAT
- Knowledge Compilation to generate the formula
- Algorithm
- Experiments
- Discussion
- Summary
Classical Planning

- States: set of **fluents variables** describing the situation
- Discrete time
- **One** initial state, goal states
- Apply action $a$
  - requires $\text{precondition}(a) \land$
  - guarantee $\text{effect}(a)$ in the next time step
Classical Planning

- States: set of **fluents variables** describing the situation
- Discrete time
- **One** initial state, goal states
- Apply action $a$
  - requires $\text{precondition}(a) \land$
  - guarantee $\text{effect}(a)$ in the next time step

Example: Robot Navigation

- State consist of fluents: horizontal position, vertical position
- Actions: move-up, move-left
Classical Planning: Complexity and Solution

- NP-complete (as SAT, exponential) assuming fixed horizon
Classical Planning: Complexity and Solution

- NP-complete (as SAT, exponential) assuming fixed horizon
- SAT solvers do well in many cases.
Classical Planning: Complexity and Solution

- NP-complete (as SAT, exponential) assuming fixed horizon
- SAT solvers do well in many cases.

- To map the *decision problem* of classical planning, horizon $k$ to SAT
  - For $k$, *generate* a propositional theory $\Phi$ *encoding* the problem
  - If $\Phi$ is SAT, report a solution
Classical Planning as SAT

- A propositional theory $\Phi$ encoding the problem, for horizon $k$
  - A variable for every action and fluent at every time step: $a_i, f_i$
  - Describe relation between actions and fluents in time
    Example: $\text{MOVE-LEFT}_1 \land \text{POS-HORIZ}_1 = 3 \supset \text{POS-HORIZ}_2 = 2$
  - Ensure that models of $\Phi$ are all the sound executions
- Call a SAT solver over $\Phi$
Classical Planning as SAT

- A propositional theory $\Phi$ encoding the problem, for horizon $k$
  - A variable for every action and fluent at every time step: $a_i, f_i$
  - Describe relation between actions and fluents in time
    Example: $\text{MOVE-LEFT}_1 \land \text{POS-HORIZ}_1=3 \supset \text{POS-HORIZ}_2=2$
  - Ensure that models of $\Phi$ are all the sound executions

- Call a SAT solver over $\Phi$

Example:

- Problem with fluents \{p, q\} and actions \{a\}
- Vars of $\Phi$ ($k = 2$): \{p_0, q_0, a_0, p_1, q_1, a_1, p_2, q_2\}
Conformant Planning SAT

- Classical planning + many possible initial states

- Logical theory \( \Phi \):
  
  same + logical description of initial states
Conformant Planning SAT

- Classical planning + many possible initial states

- Logical theory $\Phi$:
  
  same + logical description of initial states

  - Models: plans for one initial state (optimistic)
  
  - We want one plan for all initial states
    (pessimistic)
Conformant Planning SAT

- Classical planning + many possible initial states
- Logical theory $\Phi$:
  - same + logical description of initial states
  - Models: plans for one initial state (optimistic)
  - We want one plan for all initial states (pessimistic)
- Naive solution
  - Start from horizon $k = 0$, until find a solution
    - For $k$, generate a propositional theory $\Phi$
    - encoding the problem
    - Generate candidate (SAT) and Test it (SAT)
A propositional formula for Conformant Planning

- For a specific $s_0$, the plans are the models of

$$T + s_0$$

as in classical planning.
A propositional formula for Conformant Planning

- For a specific $s_0$, the plans are the models of
  $$T + s_0$$ as in classical planning
- Plans conformant for all $s_0$, are the models of?
  $$\bigwedge_{s_0 \in \text{Init}} T + s_0$$
A propositional formula for Conformant Planning

- For a specific $s_0$, the plans are the models of
  \[ T + s_0 \]
  as in classical planning

- Plans conformant for all $s_0$, are the models of?
  \[ \bigwedge_{s_0 \in \text{Init}} T + s_0 \]
  No: same plan, different executions
A propositional formula for Conformant Planning

- For a specific $s_0$, the plans are the models of
  \[ T + s_0 \] as in classical planning
- Plans conformant for all $s_0$, are the models of?
  \[ \bigwedge_{s_0 \in \text{Init}} T + s_0 \]
  No: same plan, different executions
- **Project** over actions: models of $T$ but only over actions
  \[ \text{project}(a \land b, \{a\}) = a, \quad \text{project}((a \land b) \lor c, \{a, c\}) = a \lor c \]
A propositional formula for Conformant Planning

- For a specific $s_0$, the plans are the models of
  $$T + s_0$$
as in classical planning

- Plans conformant for all $s_0$, are the models of?
  $$\bigwedge_{s_0 \in \text{Init}} T + s_0$$

  No: same plan, different executions

- **Project** over actions: models of $T$ but only over actions
  $$\text{project}(a \land b, \{a\}) = a, \quad \text{project}((a \land b) \lor c, \{a, c\}) = a \lor c$$

- **Theorem**: The conformant plans are the Models of
  $$\bigwedge_{s_0 \in \text{Init}} \text{project}[T + s_0 ; \text{Actions}]$$
Conformant Planning (horizon $k$)

1. **Generate** theory $T$ for horizon $k$

2. **Construct** the formula $T_{cf}$ where

\[ T_{cf} = \bigwedge_{s_0 \in \text{Init}} \text{project}[ T + s_0 ; \text{Actions} ] \]

3. Obtain a **Plan** by calling once a **SAT** solver over $T_{cf}$
Conformant Planning (horizon $k$)

1. **Generate** theory $T$ for horizon $k$

2. **Construct** the formula $T_{cf}$ where

   $$T_{cf} = \bigwedge_{s_0 \in \text{Init}} \text{project}[T + s_0; \text{Actions}]$$

3. Obtain a **Plan** by calling *once* a **SAT** solver over $T_{cf}$

   if we **can** do projection and conditioning $(T + s_0)$
Answer: Knowledge compilation

- **Transform** a theory to a target language, **expensive** (exponential),
  then make **cheap** operations
Answer: Knowledge compilation

- **Transform** a theory to a target language, **expensive** (exponential),
  then make **cheap** operations

- We use **deterministic - Decomposable Negation Normal Form**, $d$–DNNF, a form akin to OBDDs

- Supports **poly-time conditioning and projection**
Answer: Knowledge compilation

- **Transform** a theory to a target language, **expensive** (exponential), then make **cheap** operations
- We use **deterministic - Decomposable Negation Normal Form**, d–DNNF, a form akin to OBDDs
- Supports **poly-time conditioning and projection**
- Some OBDDs are **exponentially larger** than their equivalent d–DNNFs
- **Public libraries** for compilation from CNF to OBDDs or d–DNNFs
Conformant Planning as SAT

Start from horizon $k = 0$ increasing until find a solution

1. Generate theory $T$ for horizon $k$

2. $T$ is compiled (once) into a d–DNNF theory $T_c$

3. From $T_c$, the transformed theory

   $$T_{cf} = \bigwedge_{s_0 \in \text{init}} \text{project}[T_c + s_0 ; \text{Actions}]$$

   is obtained by linear operations in $T_c$

4. A SAT solver is called (once) over $T_{cf}$

Require: a compiler and a sat solver: no specific search algorithm
For each horizon $k$

Compile & SAT approach
For each horizon $k$

<table>
<thead>
<tr>
<th>Compile &amp; SAT approach</th>
<th>Naive approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Compile &amp; SAT approach" /> + <img src="image2" alt="Naive approach" /></td>
<td><img src="image3" alt="Naive approach" /></td>
</tr>
</tbody>
</table>
Problems

**Ring** $n$ rooms arranged in a circle. A robot can move one step a time. The room features **windows** that can be **closed** and **locked**. Initially, the position of the robot and the status of the windows is not known.

**Square Center** A robot without sensors can move in a **grid** north, south, east, and west, and its goal is to **get to the middle** of the room. The size of the grid is $2^n \times 2^n$

**Sorting networks** Build a circuit made of **compare-and-swap** gates that maps an input vector of $n$ boolean variables into the corresponding **sorted vector**
Compile time

<table>
<thead>
<tr>
<th>problem</th>
<th>$N^*$</th>
<th>CNF($T$)</th>
<th>d–DNNF $T_c$</th>
<th>CNF($T_{cf}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>vars</td>
<td>clauses</td>
<td>nodes</td>
</tr>
<tr>
<td>ring-r7</td>
<td>20</td>
<td>1081</td>
<td>3683</td>
<td>1008806</td>
</tr>
<tr>
<td>ring-r8</td>
<td>23</td>
<td>1404</td>
<td>4814</td>
<td>3887058</td>
</tr>
<tr>
<td>sq-center-e3</td>
<td>20</td>
<td>976</td>
<td>3642</td>
<td>11566</td>
</tr>
<tr>
<td>sq-center-e4</td>
<td>44</td>
<td>4256</td>
<td>16586</td>
<td>90042</td>
</tr>
<tr>
<td>sort-s7</td>
<td>16</td>
<td>1484</td>
<td>6679</td>
<td>115258</td>
</tr>
<tr>
<td>sort-s8</td>
<td>19</td>
<td>2316</td>
<td>12364</td>
<td>363080</td>
</tr>
</tbody>
</table>

- **Exponential** increasing because compilation
- **Linear** translation from d–DNNF to CNF
- Big theories do not imply **hard** problems
- Compilation is **not** the bottleneck

d–DNNF compiler by Adnan Darwiche
### Search time

<table>
<thead>
<tr>
<th>Problem</th>
<th>$N^*$</th>
<th>$#S_0$</th>
<th>sat call with horiz $N^*$</th>
<th>sc with horizon $N^* - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>time</td>
<td>decisions</td>
</tr>
<tr>
<td>Ring-r7</td>
<td>20</td>
<td>15309</td>
<td>2.1</td>
<td>2</td>
</tr>
<tr>
<td>Ring-r8</td>
<td>23</td>
<td>52488</td>
<td>$&gt;1.8\text{Gb}$</td>
<td>0</td>
</tr>
<tr>
<td>Square-center-e3</td>
<td>20</td>
<td>64</td>
<td>18.8</td>
<td>52037</td>
</tr>
<tr>
<td>Square-center-e4</td>
<td>44</td>
<td>256</td>
<td>5184.4</td>
<td>1096858</td>
</tr>
<tr>
<td>Sort-s6</td>
<td>12</td>
<td>64</td>
<td>40.0</td>
<td>34451</td>
</tr>
<tr>
<td>Sort-s7</td>
<td>16</td>
<td>128</td>
<td>3035.6</td>
<td>525256</td>
</tr>
<tr>
<td>Sort-s8</td>
<td>19</td>
<td>256</td>
<td>$&gt;2\text{h}$</td>
<td>$&gt;2\text{h}$</td>
</tr>
<tr>
<td>Square-center-e4</td>
<td>22</td>
<td>256</td>
<td>423.1</td>
<td>244085</td>
</tr>
<tr>
<td>Sort-s7</td>
<td>6</td>
<td>128</td>
<td>46.1</td>
<td>18932</td>
</tr>
<tr>
<td>Sort-s8</td>
<td>6</td>
<td>256</td>
<td>$&gt;2\text{h}$</td>
<td>$&gt;2\text{h}$</td>
</tr>
</tbody>
</table>

SAT solver: (**SIEGE_v4** or **zChaff**). Time in seconds.

**Blue**: our model-counting based planner couldn’t solve it (ICAPS’05)

---

Héctor Palacios, 2005

“Experiments”
Comparison with other works

- No many optimal conformant planners, but many suboptimal
- In general, better on very difficult problems: sort, cube
- Worst in problems close to classical planning (less uncertainty) or many objects. Ex: bomb in the toilet with 100 bombs
Discussion

- Our theories are easy to compile following their **stratified structure**: fluents $f_i$ are related with other fluents $f_i$ and actions $a_i$ and $a_{i-1}$

- Without this, compiling using the **stratification** vs. an **automatic strategy** of the compiler.
  - sort-7-ser: 12s vs 40s. Automatic: double size of the graph
  - sq-center-4: 43.9s vs $>2$ hours
Discussion

- Our theories are easy to compile following their **stratified structure**: fluents $f_i$ are related with other fluents $f_i$ and actions $a_i$ and $a_i-1$

- Without this, compiling using the **stratification** vs. an **automatic strategy** of the compiler.
  - sort-7-ser: 12s vs 40s. Automatic: double size of the graph
  - sq-center-4: 43.9s vs $>2$ hours

- Compilation **too** expensive for problems with **many** objects, but they are solved easily by others

- Other ways to project? renaming
Summary

- **Conformant Planning**: slight variation of classical planning, relevant for insight in other flavors with **uncertainty**

- **Main contribution**: propositional formula for conforman planning

- To solve a problem, **one** compiler call and **one** SAT call until $k$: optimal
  - **Simple** and powerful scheme

- **Encouraging** results

- Compilation is **not the bottleneck**

- Some instance **haven’t been** solved before (sort, cube...)

- Lot of improvement on problems close to **classical planning**
Acknowledgement

- Blai Bonet: code for parsing the PDDL problem specification and generation of CNF and previous join work
- Adnan Darwiche: compiler from CNF to d–DNNF and previous joint work
- Reviewers

thank you!
Conformant Planning Theory

Slight variation of encoding in SATPLAN

1. **Init:** a clause $C_0$ for each init clause $C \in I$.

2. **Goal:** a clause $C_N$ for each goal clause $C \in G$.

3. **Actions:** For $i = 0, 1, \ldots, N - 1$ and $a \in O$:

   $\begin{align*}
   a_i & \supset \text{pre}(a)_i \\
   \text{cond}^k(a)_i \land a_i & \supset \text{effect}^k(a)_{i+1}, \quad k = 1, \ldots, k_a
   \end{align*}$

   (preconditions)

   (effects)

4. **Frame:** for $i = 0, 1, \ldots, N - 1$, each fluent literal

   $l_i \land \bigwedge_{\text{cond}^k(a)_i \land a_i} \neg[\text{cond}^k(a)_i \land a_i] \supset l_{i+1}$

   where the conjunction ranges over the conditions $\text{cond}^k(a)_i$ associated with effects $\text{effect}^k(a)_i$ that support the complement of $l$.

5. **Exclusion:** $\neg a_i \lor \neg a'_i$ for $i = 0, \ldots, N - 1$
Conformant Planning Theory: Example

Problem:

- Fluents: $p, q, r$
- Init: $p \lor q, \neg r$. Goal: $r$
- Actions
  - $a_q$: if $p$ effect is $q$
  - $a_r$: if $q$ effect is $r$

Theory $\Phi$ for horizon $k = 2$

- Init: $p_0 \lor q_0, \neg r_0$
- Goal: $r_2$
- exclusion: $a_q 0 \otimes a_r 0$
Conformant Planning Theory: Example

Problem:
- Fluents: \( p, q, r \)
- Init: \( p \lor q, \neg r \). Goal: \( r \)
- Actions
  - \( a_q \): if \( p \) effect is \( q \)
  - \( a_r \): if \( q \) effect is \( r \)

Theory \( \Phi \) for horizon \( k = 2 \)
- Init: \( p_0 \lor q_0, \neg r_0 \)
- Goal: \( r_2 \)
- exclusion: \( a_q 0 \otimes a_r 0 \)

- effects:
  \[
  a_q 0 \land p_0 \supset q_1 \\
  a_r 0 \land q_0 \supset r_1
  \]

- frame, for each literal
  \[
  \begin{array}{c|c}
  p & p_0 \supset p_1 \\
  \neg p & \neg p_0 \supset \neg p_1 \\
  q & \neg q_0 \supset \neg q_1 \\
  \neg q & \neg(a_q 0 \land p_0) \land \neg q_0 \supset \neg q_1 \\
  r & \neg r_0 \supset \neg r_1 \\
  \neg r & \neg(a_r 0 \land r_0) \land \neg r_0 \supset \neg r_1
  \end{array}
  \]

etc.
**deterministic - Decomposable Negation Normal Form (d–DNNF)**

- Normal form: NNF satisfying determinism and decomposability (see paper for details)
  - **Deterministic**: for each AND node, no variable appears in more than one conjunct
  - **Decomposable**: for each OR node, disjuncts are pairwise logically inconsistent
- Compiling to d–DNNF: a naive algorithm proceed doing **exhaustive** DPLL (all SAT)
- d-DNNF compilations are, typically, **exponentially** bigger
- Projection and conditioning are **lineal** in the size of the d-DNNF
**d-DNNF: Example**

Theory

\[ a \lor \neg a \]
\[ c \lor d \]
\[ \neg c \lor b \]

- **Decomposable?**
  
  For each OR node, disjuncts are pairwise logically inconsistent

- **Deterministic?**
  
  For each AND node, no variable appears in more than one conjunct
Calculating the CNF efficiently

- We can ask the compiler to give the d–DNNF
  - Projected over actions and \( \text{vars}(s_0) \) (no fluents \( i > 0 \))
  - Make cases analysis **first** over \( \text{vars}(s_0) \)

- Then project \( [T + s_0; \text{Actions}] \) can be **extracted as a subgraph**

Then, we can construct \( \bigwedge_{s_0 \in \text{init}} \text{project}[T + s_0; \text{Actions}] \) by making a **new graph** with the extracted subgraphs. Easy to CNF!
• Fluents: $p, q, r$

• Init: $p \lor q, \neg r$. Goal: $r$

• Actions:
  
  – $a_q$: if $p$ effect is $q$
  
  – $a_r$: if $q$ effect is $r$

• Solution: $a_q, a_r$

Compiling for $k = 2$ ...
• Fluents: $p, q, r$

• Init: $p \lor q, \neg r$. Goal: $r$

• Actions:
  - $a_q$: if $p$ effect is $q$
  - $a_r$: if $q$ effect is $r$

• Solution: $a_q, a_r$

Compiling for $k = 2$ ...

Asking the compiler to:

• Make cases analysis \textbf{first} over init vars: $p_0, q_0, r_0$

• \textbf{Project while compiling} over init + action vars
Fluents: \( p, q, r \)

Init: \( p \lor q, \neg r \). Goal: \( r \)

Actions:
- \( a_q \): if \( p \) effect is \( q \)
- \( a_r \): if \( q \) effect is \( r \)

Solution: \( a_q, a_r \)

Compiling for \( k = 2 \) ...

Asking the compiler to:
- Make cases analysis first over init vars: \( p_0, q_0, r_0 \)
- Project while compiling over init + action vars
Projection, a logical operation

- Don’t want to care about some variables
Projection, a logical operation

- Don’t want to care about some variables
- Example: want to *forget* \( f_1 \) from \( \phi = (a_1 \land f_1) \lor a_2 \)
Projection, a logical operation

- Don’t want to care about some variables

- Example: want to forget $f_1$ from $\phi = (a_1 \land f_1) \lor a_2$

  $\text{project}[\phi; \{a_1, a_2\}] = \exists f_1 \phi$

  $= (\phi \upharpoonright f_1 = \text{true}) \lor (\phi \upharpoonright f_1 = \text{false})$

  $= (((a_1 \land \text{true}) \lor a_2) \lor ((a_1 \land \text{false}) \lor a_2))$

  $= (a_1 \lor a_2)$

Models of $\phi = (a_1 \land f_1) \lor a_2$, if we don’t care about $f_1$, are the models of $a_1 \lor a_2$
Projection, a logical operation

- Don’t want to care about some variables

- Example: want to forget $f_1$ from $\phi = (a_1 \land f_1) \lor a_2$

$$\text{project} \left[ \phi; \{a_1, a_2\} \right] = \exists f_1 \phi$$

$$= (\phi \mid f_1 = \text{true}) \lor (\phi \mid f_1 = \text{false})$$

$$= ((a_1 \land \text{true}) \lor a_2) \lor ((a_1 \land \text{false}) \lor a_2)$$

$$= (a_1 \lor a_2)$$

Models of $\phi = (a_1 \land f_1) \lor a_2$, if we don’t care about $f_1$, are the models of $a_1 \lor a_2$

- The projection of a formula over a subset of its variables is the strongest formula over those variables
Discussion (2)

- Conformant Planning can be solved as a QBF of the form solve

\[ \exists Plan \ \forall s_0 \ \exists \text{execution} \ \ T \]

Our method is **simple and generic**. Can be used to solve QBFs?

- Our CNFs theories are probably the biggest compiled to d-DNNF. Can we detect **stratified** structure in other CNFs?

- Relation with other problems that can’t be map to SAT: all solutions to CNFs, unsat of CNFs, weighted CNF, maxSAT, MPE (Bay Nets).

- Further work: new theoretical notions for understanding the gap between theory and practice in SAT and CSP and beyond them: hypertree decomposition (chen & dalmau), semantic width (dechter), strong backdoors (gomes, selman).