Pruning Conformant Plans by Counting Models on Compiled d-DNNF Representations

H. Palacios  B. Bonet  A. Darwiche  H. Geffner
UPF      USB      UCLA      ICREA/UPF
Introduction
Motivation

- Planners in the classical setting built around two notions: **branching** and **pruning**.
Motivation

- Planners in the classical setting built around two notions: **branching** and **pruning**.

- In search-based approaches:
  - branching is directional (forward or backward),
  - pruning by comparison of estimated costs (heuristics).
Motivation

- Planners in the classical setting built around two notions: **branching** and **pruning**.

- In search-based approaches:
  - branching is directional (forward or backward),
  - pruning by comparison of estimated costs (heuristics).

- In SAT-based approaches:
  - branching is non-directional (instantiation of variables),
  - pruning by unit resolution and clause learning.
Motivation

- Planners in the classical setting built around two notions: **branching** and **pruning**.

- In search-based approaches:
  - branching is directional (forward or backward),
  - pruning by comparison of estimated costs (heuristics).

- In SAT-based approaches:
  - branching is non-directional (instantiation of variables),
  - pruning by unit resolution and clause learning.

- In this work, we introduce a branch-and-prune scheme for conformant planning, based on model counting operations implemented in linear time over compiled representations of the problem.
Conformant vs Classical Planning

- Conformant planning involves non-deterministic transitions and sets of possible initial states
Conformant vs Classical Planning

- Conformant planning involves non-deterministic transitions and sets of possible initial states

- A conformant plan must work for every possible initial state and transition
Conformant vs Classical Planning

- Conformant planning involves non-deterministic transitions and sets of possible initial states

- A conformant plan must work for every possible initial state and transition

- Unlike classical planning, conformant planning cannot be reduced to model finding over a logical encoding
Conformant vs Classical Planning

- Conformant planning involves non-deterministic transitions and sets of possible initial states

- A conformant plan must work for every possible initial state and transition

- Unlike classical planning, conformant planning cannot be reduced to model finding over a logical encoding

- Indeed, a model $M$ for a planning theory represents an “optimistic” plan, a plan that works for some initial states, but not necessarily all
Testing If a Plan is Conformant

- If all actions are deterministic, it is simple to check whether a plan $A$ (full action valuation) is conformant:

$$A \text{ is conformant} \iff \# \text{Models(Theory + } A) = \# \text{ init. states}$$
Testing If a Plan is Conformant

- If all actions are deterministic, it is simple to check whether a plan $A$ (full action valuation) is conformant:

$$A \text{ is conformant } \iff \text{#Models(Theory } + A \text{) = # init. states}$$

- Model counting is hard (#P-complete), yet it can be done efficiently if the theory is in suitable form
Testing If a Plan is Conformant

- If all actions are deterministic, it is simple to check whether a plan $A$ (full action valuation) is conformant:

$$A \text{ is conformant } \iff \#\text{Models}(\text{Theory} + A) = \# \text{ init. states}$$

- Model counting is hard ($\#P$-complete), yet it can be done efficiently if the theory is in suitable form

- Our goal, however, is not only to check whether a plan is conformant but to find one such plan
First approach: generate-and-test ... too inefficient 😞
Finding Conformant Plans

- First approach: generate-and-test ... too inefficient 😞

- Better: generate plans incrementally, pruning those that cannot lead to conformant plans:
  - Start with an empty plan $A$
  - Extend $A$ by picking and instantiating action variables
  - Prune $A$ if cannot lead to a conformant plan
Finding Conformant Plans

- First approach: generate-and-test ... too inefficient 😞

- Better: generate plans incrementally, pruning those that cannot lead to conformant plans:
  - Start with an empty plan $A$
  - Extend $A$ by picking and instantiating action variables
  - Prune $A$ if cannot lead to a conformant plan

- **Key Question:** how to detect that partial plan cannot lead to conformant plan?
We’ll need a second logical operation: **projection** which is dual of variable elimination (existential quantification)
Pruning Plans

- We’ll need a second logical operation: projection which is dual of variable elimination (existential quantification)

- The projection of $T$ on subset $V$ of vars is the strongest theory $T'$ over $V$ that is logically implied by $T$; e.g.
  - $Proj((x \lor y) \land z, \{x, y\}) = x \lor y$
  - $Proj((x \lor y) \land z, \{z\}) = z$
  - $Proj((x \lor y) \land z, \{x\}) = true$
Pruning Plans

- We'll need a second logical operation: projection which is dual of variable elimination (existential quantification)

- The projection of $T$ on subset $V$ of vars is the strongest theory $T'$ over $V$ that is logically implied by $T$; e.g.
  - $Proj((x \lor y) \land z, \{x, y\}) = x \lor y$
  - $Proj((x \lor y) \land z, \{z\}) = z$
  - $Proj((x \lor y) \land z, \{x\}) = true$

- Partial plan $A$ can be pruned if

  $\#\text{Models}(Proj(\text{Theory} + A, \text{init vars})) \neq \# \text{init. states}$

  i.e. $A$ won't work for some initial state!
Pruning Plans

- We’ll need a second logical operation: projection which is dual of variable elimination (existential quantification)

- The projection of $T$ on subset $V$ of vars is the strongest theory $T'$ over $V$ that is logically implied by $T$; e.g.
  - $\text{Proj}((x \lor y) \land z, \{x, y\}) = x \lor y$
  - $\text{Proj}((x \lor y) \land z, \{z\}) = z$
  - $\text{Proj}((x \lor y) \land z, \{x\}) = \text{true}$

- Partial plan $A$ can be pruned if
  \[
  \#\text{Models}(\text{Proj}(\text{Theory} + A, \text{init vars})) \neq \# \text{init. states}
  \]
  I.e. $A$ won’t work for some initial state!

- **Key Point:** efficient implementation of #Models and $\text{Proj}$ if theory is in d-DNNF format (a generalization of OBDDs)
Contribution

- A conformant, logic-based, branch-and-prune planner

- Prunes partial plans based on project and model counting operations.

- which are supported in linear in d-DNNFs

- Approach very flexible; e.g.
  - Can accommodate arbitrary goals
  - generate plans that conform with $X\%$ of initial states
  - can maximize “conformity” if no plan is 100% conformant

- Performance is good; although lots of room for improvement and variations

- Resulting plans are optimal in number of steps
Conformant Planning
**Problem**: \( P = \langle F, O, I, G \rangle \)
- fluent symbols \( F \),
- *deterministic* actions \( a \in O \) defined by preconditions \( prec(a) \) and conditional effects \( c^k(a) \rightarrow e^k(a), k = 1 \ldots n_a \),
- \( I, G \) descriptions of initial and goal situations.
Formulation and Encoding

- **Problem:** \( P = \langle F, O, I, G \rangle \)
  - fluent symbols \( F \),
  - *deterministic* actions \( a \in O \) defined by preconditions \( \text{prec}(a) \) and conditional effects \( c^k(a) \rightarrow e^k(a), k = 1 \ldots n_a \),
  - \( I, G \) descriptions of initial and goal situations.

- For a given plan horizon \( N \), the problem \( P \) is encoded as a CNF theory \( T(P) \) whose size is polynomial in the size of \( P \).
Problem: \( P = \langle F, O, I, G \rangle \)
- fluent symbols \( F \),
- deterministic actions \( a \in O \) defined by preconditions \( \text{pre}(a) \) and conditional effects \( c^k(a) \rightarrow e^k(a), k = 1 \ldots n_a \),
- \( I, G \) descriptions of initial and goal situations.

For a given plan horizon \( N \), the problem \( P \) is encoded as a CNF theory \( T(P) \) whose size is polynomial in the size of \( P \).

In the classical setting, there is one-one correspondence between models of \( T(P) \) and plans of length \( N \), and thus planning can be reduced to model finding.
Validity

- **Partial Plans:**
  - Collection of action literals denoted by $T_A$
  - Complete if it mentions all action literals
Validity

- **Partial Plans:**
  - Collection of action literals denoted by $T_A$
  - Complete if it mentions all action literals

- **Validity:** a partial plan $T_A$ is **valid** iff for each initial state $s$ the formulas $T_A \land T(P) \land s$ is consistent.
Validity

- **Partial Plans:**
  - Collection of action literals denoted by $T_A$
  - Complete if it mentions all action literals

- **Validity:** a partial plan $T_A$ is **valid** iff for each initial state $s$
  the formulas $T_A \land T(P) \land s$ is consistent.

- Two important properties:
  - A complete plan that is valid is conformant
  - An invalid partial plan cannot lead to a conformant plan
Validity as Model Count and Projection

- Partial plan $T_A$ valid if

$$\#\text{Models}(\text{Proj}(T(P) + T_A, F_0)) = \#\text{Models}(T_0(P))$$

where $T_0(P)$ is the set of clauses for initial situation, and $F_0$ is the set of fluents at time $t = 0$ (init)
Validity as Model Count and Projection

- Partial plan $T_A$ valid if
  
  $\#\text{Models}(\text{Proj}(T(P) + T_A, F_0)) = \#\text{Models}(T_0(P))$

  where $T_0(P)$ is the set of clauses for initial situation, and $F_0$ is the set of fluents at time $t = 0$ (init)

- **Key Issue:** how to perform Model Count and Projection efficiently in every node $A$ of the search tree?
Deterministic and Decomposable
Negation Normal Forms
A propositional sentence is in NNF if it’s constructed from literals using only conjunctions and disjunctions;
Negation Normal Forms

- A propositional sentence is in NNF if it's constructed from literals using only conjunctions and disjunctions;

- Represented by a rooted DAG whose leaves are labeled with literals, TRUE or FALSE, and its internal nodes are labeled with conjunction or disjunction;
A NNF is **decomposable** if no variable appears in more than one conjunct for each conjunction node;
Decomposable and Deterministic NNFs

- A NNF is **decomposable** if no variable appears in more than one conjunct for each conjunction node;

- A NNF is **deterministic** if the disjuncts of each disjunction node are pairwise logically inconsistent;
Decomposable and Deterministic NNFs

- A NNF is **decomposable** if no variable appears in more than one conjunct for each conjunction node;

- A NNF is **deterministic** if the disjuncts of each disjunction node are pairwise logically inconsistent;

- A d-DNNF (Darwiche 2001) supports a number of operations
  - satisfiability,
  - clause entailment,
  - model counting,
  - (restricted) projection,
  - etc.
  in linear time in the size of the NNF.
Compiling Theories into d-DNNF

- Compiling theories into d-DNNF is NP-hard but no harder than compiling into OBDDs

- Indeed, OBDDs can be efficiently translated into d-DNNFs; but not the other way around

- d-DNNF compilers exploit decomposition, unit resolution, dynamic variable ordering, etc.

- In proposed planner, first step is to compile CNF theory into d-DNNF
The Conformant Planner
- **Preprocessing:** a problem $P$ and horizon $N$ is translated into a CNF theory $T(P)$ and then compiled into a d-DNNF $T$.

- **Branching:** at a node $n$ in the search tree, VPLAN branches by selecting an uninstantiated *action* literal.

- **Pruning:** a node $n$ is pruned when the d-DNNF theory $T_n$ associated with $n$ fails the *validity test* implemented with model counting and projection over the compiled theory.
Experimental Results
Benchmark

- Problems:
  - Ring: lock and close windows
  - Sorting Networks: circuit synthesis
  - Square/Cube Center: navigation problem
  - Blocks: conformant version of blocksworld

- Non-trivial problems, only **optimal** planner that can handle all of them is (Rintanen 2004).
### Compilation

<table>
<thead>
<tr>
<th>Problem</th>
<th>$N^*$</th>
<th>CNF theory</th>
<th>d-DNNF theory</th>
<th>Time/Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocks-2</td>
<td>2</td>
<td>34</td>
<td>105</td>
<td>0.03/0.06</td>
</tr>
<tr>
<td>blocks-3</td>
<td>9</td>
<td>444</td>
<td>2913</td>
<td>0.25/1.13</td>
</tr>
<tr>
<td>blocks-4</td>
<td>26</td>
<td>3036</td>
<td>40732</td>
<td>77.5/752.65</td>
</tr>
<tr>
<td>sq-center-2</td>
<td>8</td>
<td>200</td>
<td>674</td>
<td>0.1/0.39</td>
</tr>
<tr>
<td>sq-center-3</td>
<td>20</td>
<td>976</td>
<td>3642</td>
<td>0.7/6.7</td>
</tr>
<tr>
<td>sq-center-4</td>
<td>44</td>
<td>4256</td>
<td>16586</td>
<td>31.17/512.54</td>
</tr>
<tr>
<td>ring-3</td>
<td>8</td>
<td>209</td>
<td>669</td>
<td>0.11/0.48</td>
</tr>
<tr>
<td>ring-4</td>
<td>11</td>
<td>364</td>
<td>1196</td>
<td>0.62/2.52</td>
</tr>
<tr>
<td>ring-5</td>
<td>14</td>
<td>561</td>
<td>1874</td>
<td>3.68/16.4</td>
</tr>
<tr>
<td>ring-6</td>
<td>17</td>
<td>800</td>
<td>2703</td>
<td>23.77/120.58</td>
</tr>
<tr>
<td>ring-7</td>
<td>20</td>
<td>1081</td>
<td>3683</td>
<td>221.58/1096.7</td>
</tr>
<tr>
<td>ring-8</td>
<td>23</td>
<td>1404</td>
<td>4814</td>
<td>2018.32/12463.3</td>
</tr>
<tr>
<td>sortnet-3</td>
<td>3</td>
<td>51</td>
<td>122</td>
<td>0.03/0.09</td>
</tr>
<tr>
<td>sortnet-4</td>
<td>5</td>
<td>150</td>
<td>409</td>
<td>0.04/0.19</td>
</tr>
<tr>
<td>sortnet-5</td>
<td>9</td>
<td>420</td>
<td>1343</td>
<td>0.51/1.4</td>
</tr>
<tr>
<td>sortnet-6</td>
<td>12</td>
<td>813</td>
<td>3077</td>
<td>1.28/7.12</td>
</tr>
<tr>
<td>sortnet-7</td>
<td>16</td>
<td>1484</td>
<td>6679</td>
<td>8.29/56.61</td>
</tr>
<tr>
<td>sortnet-8</td>
<td>19</td>
<td>2316</td>
<td>12364</td>
<td>56.73/427.58</td>
</tr>
<tr>
<td>sortnet-9</td>
<td>25</td>
<td>3870</td>
<td>24414</td>
<td>780.77/6316.53</td>
</tr>
</tbody>
</table>
## Search

<table>
<thead>
<tr>
<th>problem</th>
<th>$N^*$</th>
<th>$#S_0$</th>
<th>search at horizon $k$</th>
<th>search at horizon $k - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>time</td>
<td>backtracks</td>
</tr>
<tr>
<td>blocks-2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>blocks-3</td>
<td>9</td>
<td>13</td>
<td>0.02</td>
<td>7</td>
</tr>
<tr>
<td>blocks-4</td>
<td>26</td>
<td>73</td>
<td>&gt; 2h</td>
<td>&gt; 76029</td>
</tr>
<tr>
<td>sq-center-2</td>
<td>8</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sq-center-3</td>
<td>20</td>
<td>64</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>sq-center-4</td>
<td>44</td>
<td>256</td>
<td>&gt; 2h</td>
<td>&gt; 188597</td>
</tr>
<tr>
<td>ring-3</td>
<td>8</td>
<td>81</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ring-4</td>
<td>11</td>
<td>324</td>
<td>0.06</td>
<td>1</td>
</tr>
<tr>
<td>ring-5</td>
<td>14</td>
<td>1215</td>
<td>0.71</td>
<td>2</td>
</tr>
<tr>
<td>ring-6</td>
<td>17</td>
<td>4374</td>
<td>3.49</td>
<td>4</td>
</tr>
<tr>
<td>ring-7</td>
<td>20</td>
<td>15309</td>
<td>24.48</td>
<td>5</td>
</tr>
<tr>
<td>ring-8</td>
<td>23</td>
<td>52488</td>
<td>128.64</td>
<td>7</td>
</tr>
<tr>
<td>sortnet-3</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sortnet-4</td>
<td>5</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sortnet-5</td>
<td>9</td>
<td>32</td>
<td>0.02</td>
<td>0</td>
</tr>
<tr>
<td>sortnet-6</td>
<td>12</td>
<td>64</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>sortnet-7</td>
<td>16</td>
<td>128</td>
<td>&gt; 2h</td>
<td>&gt; 102300</td>
</tr>
</tbody>
</table>
Wrap Up

● Contribution
● Interesting

Thanks
Contribution

- A conformant, logic-based, branch-and-prune planner
- Prunes partial plans based on project and model counting operations ..
- which are supported in linear in d-DNNFs
- Approach very flexible; e.g.
  - Can accommodate arbitrary goals
  - generate plans that conform with 90% of initial states
  - “maximizes” conformant if there is no 100% conformant plans
- Performance is good; although lots of room for improvement and variations
- Resulting plans are optimal
Current bottleneck is not compilation but search
- Current bottleneck is not compilation but search

- If CNF is compiled following certain variable order, the search can be done **backtrack free**
Interesting

- Current bottleneck is not compilation but search

- If CNF is compiled following certain variable order, the search can be done backtrack free

- However, this doesn’t work in practice
Interesting

- Current bottleneck is not compilation but search

- If CNF is compiled following certain variable order, the search can be done **backtrack free**

- However, this doesn’t work in practice

- Interesting to study further the tradeoff compilation vs search
Thanks. Questions ...