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

H. Palacios — UPF

B. Bonet — USB

A. Darwiche — UCLA

H. Geffner — 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 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:

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

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 is conformant $\iff$ #Models(Theory $+ A) =$ # 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**

Finding Conformant Plans

- 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?

Pruning Plans

- 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.
  - $Proj((x \vee y) \wedge z, \{x, y\}) = x \vee y$
  - $Proj((x \vee y) \wedge z, \{z\}) = z$
  - $Proj((x \vee y) \wedge z, \{x\}) =$ 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.
  - $Proj((x \vee y) \wedge z, \{x, y\}) = x \vee y$
  - $Proj((x \vee y) \wedge z, \{z\}) = z$
  - $Proj((x \vee y) \wedge z, \{x\}) = \text{true}$
- 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.
  - $Proj((x \vee y) \wedge z, \{x, y\}) = x \vee y$
  - $Proj((x \vee y) \wedge z, \{z\}) = z$
  - $Proj((x \vee y) \wedge z, \{x\}) =$ 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!
- **Key Point:** efficient implementation of #Models and $Proj$ 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

Formulation and Encoding

- **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 $prec(a)$ 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

Formulation and Encoding

- **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.
- 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 $T_A \wedge T(P) \wedge s$ 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 $T_A \wedge T(P) \wedge s$ 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

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

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

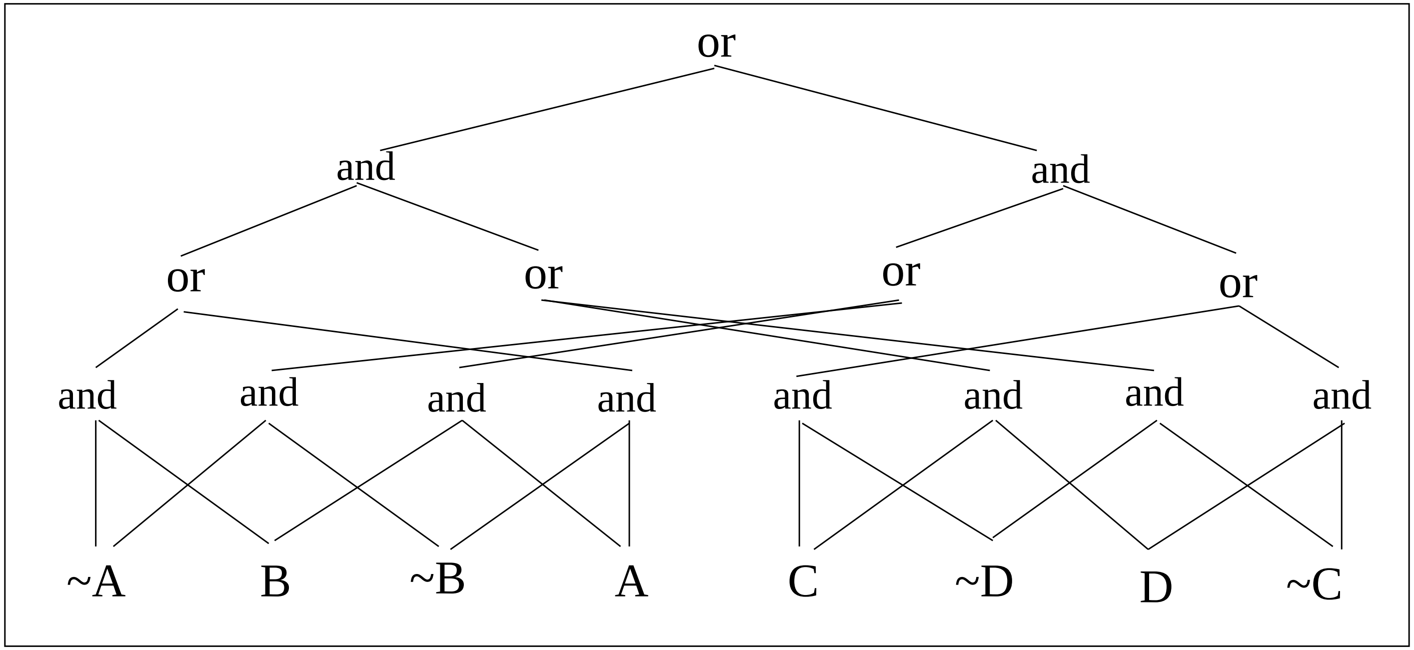
$$\#\text{Models}(Proj(T(P) + T_A, F_0)) \;=\; \#Models(T_0(P))$$

  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?

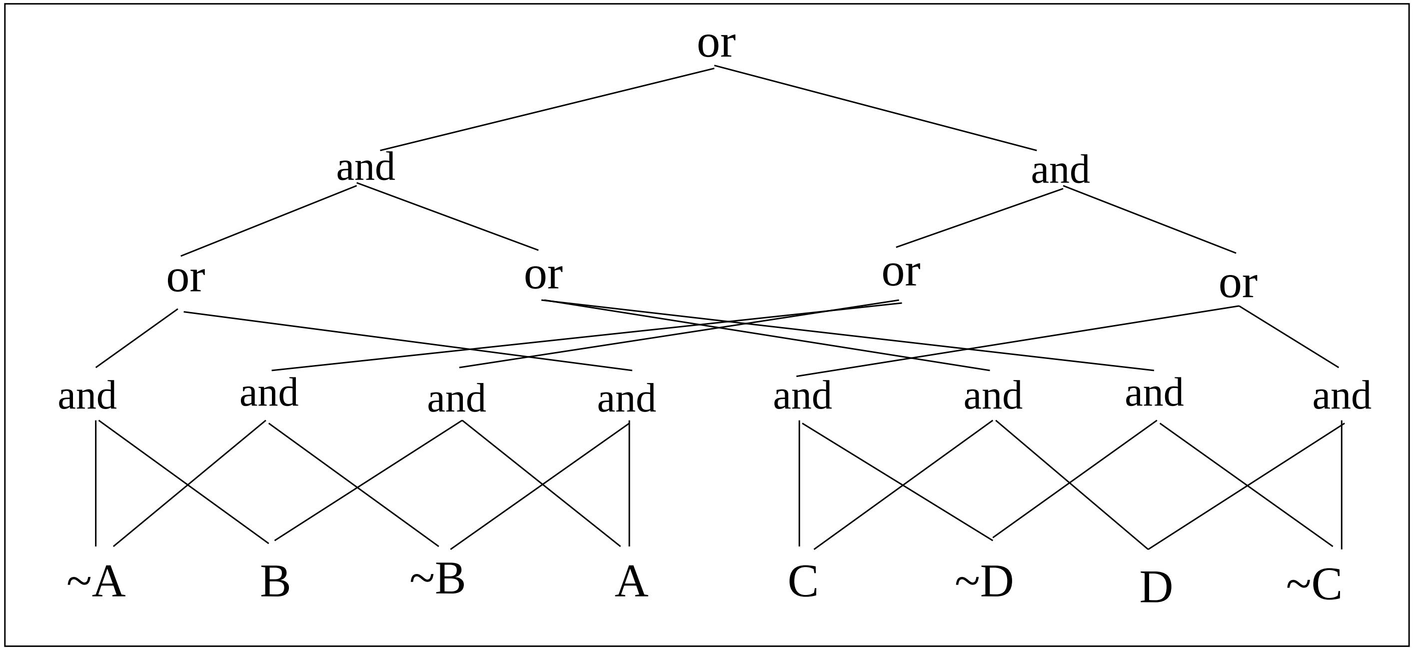
Deterministic and Decomposable Negation Normal Forms

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;

Decomposable and Deterministic NNFs

- 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

VPLAN

- **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

		CNF theory		d-DNNF theory		
problem	N^*	vars	clauses	nodes	edges	time/acc
blocks-2	2	34	105	61	97	0.03/0.06
blocks-3	9	444	2913	4672	20010	0.25/1.13
blocks-4	26	3036	40732	225396	913621	77.5/752.65
sq-center-2	8	200	674	1000	2216	0.1/0.39
sq-center-3	20	976	3642	9170	19555	0.7/6.7
sq-center-4	44	4256	16586	79039	164191	31.17/512.54
ring-3	8	209	669	2753	6161	0.11/0.48
ring-4	11	364	1196	13239	29295	0.62/2.52
ring-5	14	561	1874	60338	132045	3.68/16.4
ring-6	17	800	2703	254379	551641	23.77/120.58
ring-7	20	1081	3683	1018454	2195393	221.58/1096.7
ring-8	23	1404	4814	3928396	8406323	2018.32/12463.3
sortnet-3	3	51	122	133	230	0.03/0.09
sortnet-4	5	150	409	1048	2325	0.04/0.19
sortnet-5	9	420	1343	7395	17823	0.51/1.4
sortnet-6	12	813	3077	30522	77015	1.28/7.12
sortnet-7	16	1484	6679	116138	294840	8.29/56.61
sortnet-8	19	2316	12364	369375	931097	56.73/427.58
sortnet-9	25	3870	24414	1264508	3075923	780.77/6316.53

Search

problem	N^*	$\#S_0$	search at horizon k time	backtracks	#act	search at horizon $k-1$ time	backtracks
blocks-2	2	3	0	1	2	0	1
blocks-3	9	13	0.02	7	9	144.45	248619
blocks-4	26	73	$>$ 2h	$>$ 76029		$>$ 2h	$>$ 78714
sq-center-2	8	16	0	0	8	0.02	243
sq-center-3	20	64	0.05	0	20	$>$ 2h	$>$ 3741672
sq-center-4	44	256	$>$ 2h	$>$ 188597		$>$ 2h	$>$ 191030
ring-3	8	81	0	0	8	0	5
ring-4	11	324	0.06	1	11	0.02	5
ring-5	14	1215	0.71	2	14	0.16	5
ring-6	17	4374	3.49	4	17	0.69	5
ring-7	20	15309	24.48	5	20	3.35	5
ring-8	23	52488	128.64	7	23	13.08	5
sortnet-3	3	8	0	0	3	0	5
sortnet-4	5	16	0	0	5	0.05	421
sortnet-5	9	32	0.02	0	9	$>$ 2h	$>$ 4845305
sortnet-6	12	64	0.2	1	12	$>$ 2h	$>$ 458912
sortnet-7	16	128	$>$ 2h	$>$ 102300		$>$ 2h	$>$ 104674

Wrap Up

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

Interesting

- Current bottleneck is not compilation but search

Interesting

- 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 ...