This is a solution! (... but is it though?)
Verifying solutions of hierarchical planning problems

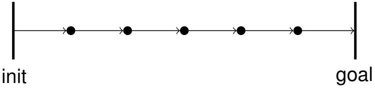
Gregor Behnke, Daniel Höller, Susanne Biundo

Ulm University, Institute of Artificial Intelligence

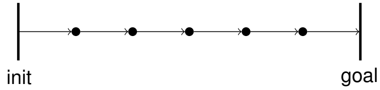
June 21, 2016

ICAPS 2017 – Pittsburgh

Plan Verification

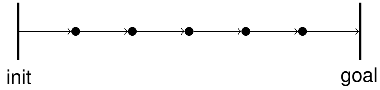


Plan Verification



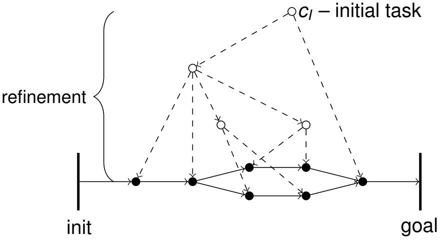
- $\mathcal{O}(n)$ for totally ordered classical plans

Plan Verification



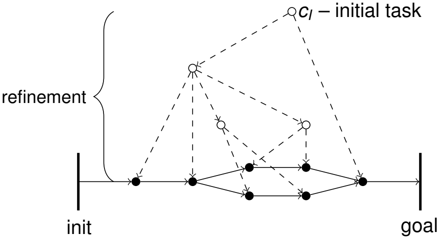
- $\mathcal{O}(n)$ for totally ordered classical plans
- VAL provides for plan verification in classical domains

Plan Verification



- $\mathcal{O}(n)$ for totally ordered classical plans
- VAL provides for plan verification in classical domains
- $\mathbb{NP}$-complete for HTN (Hierarchical Task Network) planning [Behnke et al. 2015]

Plan Verification



- $\mathcal{O}(n)$ for totally ordered classical plans
- VAL provides for plan verification in classical domains
- $\mathbb{NP}$-complete for HTN (Hierarchical Task Network) planning [Behnke et al. 2015]
- So far, no HTN plan verifier exists

Why plan verification?

Plan Verification can be used for

Why plan verification?

Plan Verification can be used for

- validating HTN planners

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions
- plan repair

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions
- plan repair

What have we done?

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions
- plan repair

What have we done?

1. Provided a translation of Plan Verification problem into SAT

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions
- plan repair

What have we done?

1. Provided a translation of Plan Verification problem into SAT
2. Provided succinct decomposition depth bounds for plans

Why plan verification?

Plan Verification can be used for

- validating HTN planners
- HTN planning competitions (. . . future work)
- post-optimisation of solutions
- plan repair

What have we done?

1. Provided a translation of Plan Verification problem into SAT
2. Provided succinct decomposition depth bounds for plans
3. Showed that verifying plans using SAT is empirically feasible

Hierarchical Task Network (HTN) Planning

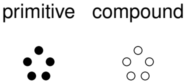
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation,
  executable from the initial state

Hierarchical Task Network (HTN) Planning



$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning

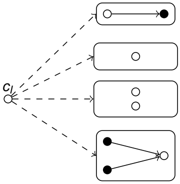
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



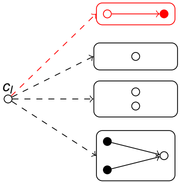
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



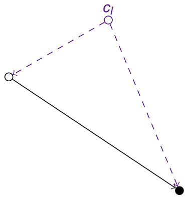
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



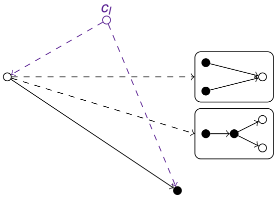
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



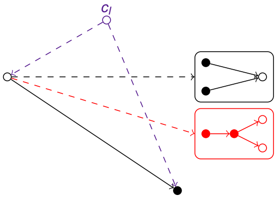
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



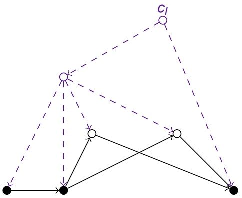
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



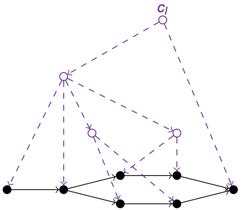
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



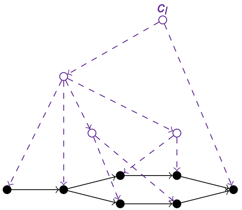
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



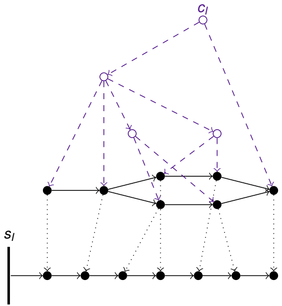
$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Hierarchical Task Network (HTN) Planning



$\mathcal{P} = (P, C, c_I, M, L, s_I)$

- P a set of primitive tasks
- C a set of compound tasks
- $c_I \in C$ the initial task
- $M \subseteq C \times 2^{TN}$ the methods
- L a set of variables
- $s_I \subseteq L$ the initial state

A solution $tn \in Sol(\mathcal{P})$ must

- be a refinement of the initial task
- only contain primitive tasks
- have a linearisation, executable from the initial state

Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

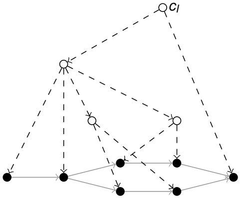
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement



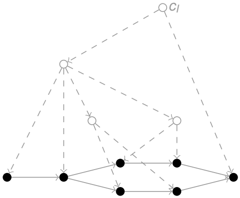
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement
- primitive



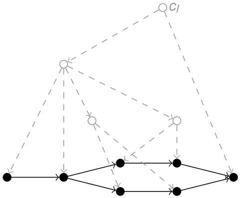
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement
- primitive
- executability



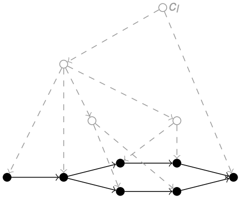
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

What do we have to check?

- refinement
- primitive
- executability



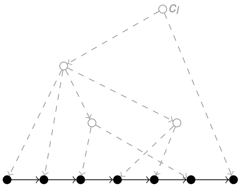
Plan Verification

Definition (VERIFYTN)

Let $\mathcal{P}$ be a planning problem and tn be a task network.
Decide whether $\mathrm{tn} \in \mathit{Sol}(\mathcal{P})$.

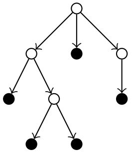
What do we have to check?

- refinement
- primitive
- executability



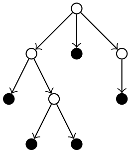
Translation into SAT

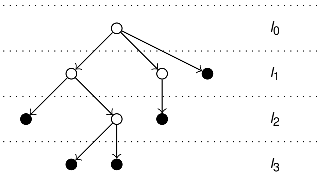
- Let's have a look at a Decomposition Tree leading to a solution



Translation into SAT

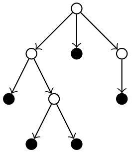
- Let's have a look at a Decomposition Tree leading to a solution
- We can arrange its vertices (i.e. primitive and abstract tasks) in layers

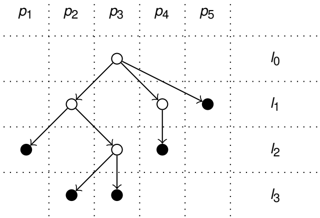




Translation into SAT

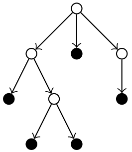
- Let's have a look at a Decomposition Tree leading to a solution
- We can arrange its vertices (i.e. primitive and abstract tasks) in layers
- ... and assign each vertex a row.

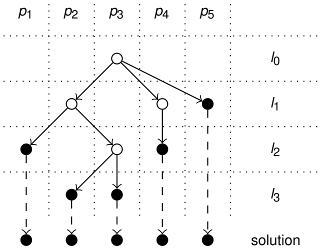




Translation into SAT

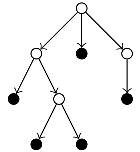
- Let's have a look at a Decomposition Tree leading to a solution
- We can arrange its vertices (i.e. primitive and abstract tasks) in layers
- ... and assign each vertex a row.

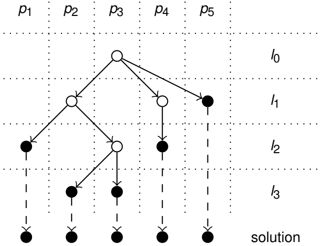




Translation into SAT

- Let's have a look at a Decomposition Tree leading to a solution
- We can arrange its vertices (i.e. primitive and abstract tasks) in layers
- ... and assign each vertex a row.
- Our SAT formula models this assignment process





Translation into SAT

Clauses describe local restrictions at position

Translation into SAT

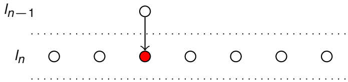
Clauses describe local restrictions at position

- Node Constraints
    - at most one task

Translation into SAT

Clauses describe local restrictions at position

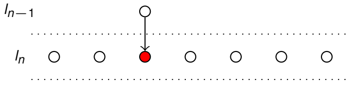
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer



Translation into SAT

Clauses describe local restrictions at position

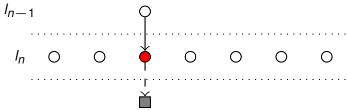
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent



Translation into SAT

Clauses describe local restrictions at position

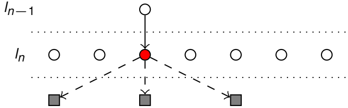
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method



Translation into SAT

Clauses describe local restrictions at position

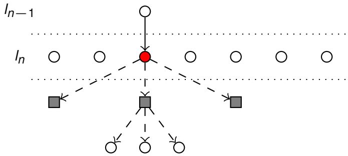
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method



Translation into SAT

Clauses describe local restrictions at position

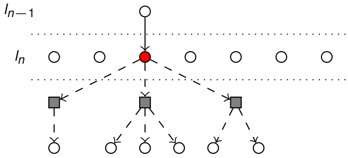
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks



Translation into SAT

Clauses describe local restrictions at position

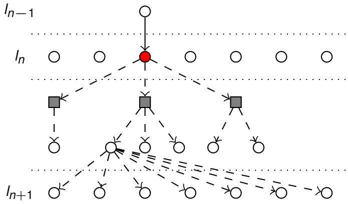
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks



Translation into SAT

Clauses describe local restrictions at position

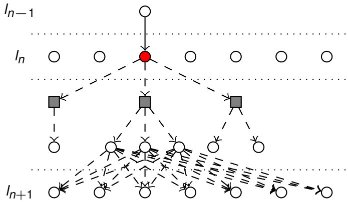
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer



Translation into SAT

Clauses describe local restrictions at position

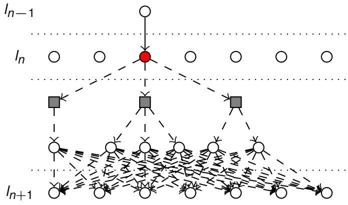
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer



Translation into SAT

Clauses describe local restrictions at position

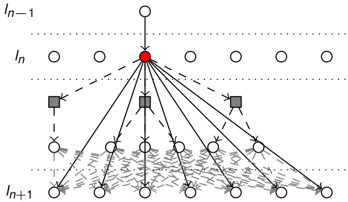
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer



Translation into SAT

Clauses describe local restrictions at position

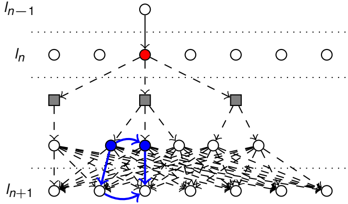
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer
  - subtasks are children of parent



Translation into SAT

Clauses describe local restrictions at position

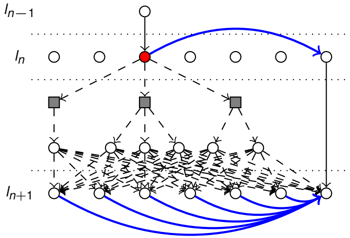
- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer
  - subtasks are children of parent
  - subtasks must respect the method's order



Translation into SAT

Clauses describe local restrictions at position

- Node Constraints
  - at most one task
- Parent Constraints
  - only one parent in previous layer
  - no task if no parent
- Children Constraints for every method
  - if abstract, exactly one method
  - selected method must have subtasks
  - subtasks must occur in the next layer
  - subtasks are children of parent
  - subtasks must respect the method's order
  - subtasks must respect parent's order



Bounding Decomposition Height

- The translation assumes a height parameter K

Bounding Decomposition Height

- The translation assumes a height parameter K
- To be correct, we need to determine K, s.t. every plan of length n has a decomposition of height $\leq K$ or none at all

Bounding Decomposition Height

- The translation assumes a height parameter K

- To be correct, we need to determine K, s.t. every plan of length n has a decomposition of height $\leq K$ or none at all
- I.e. we need to compute the maximum depth of a decomposition that can lead to a plan of length n

Bounding Decomposition Height

- The translation assumes a height parameter K
- To be correct, we need to determine K, s.t. every plan of length n has a decomposition of height $\leq K$ or none at all
- I.e. we need to compute the maximum depth of a decomposition that can lead to a plan of length n
- We have developed four methods to compute an upper bound for K

Bounding Decomposition Height

- The translation assumes a height parameter K
- To be correct, we need to determine K, s.t. every plan of length n has a decomposition of height $\leq K$ or none at all
- I.e. we need to compute the maximum depth of a decomposition that can lead to a plan of length n
- We have developed four methods to compute an upper bound for K
  The first three are described in the paper, the fourth is new.

Bounding Decomposition Height

Method 1:

- In Behnke et al. (ICAPS 2015) we showed that plan verification is $\mathbb{NP}$ complete
- The proof provides a theoretical upper bound $K_{theo} = 2|plan|(|C| + 1)$

Bounding Decomposition Height

Method 1:

- In Behnke et al. (ICAPS 2015) we showed that plan verification is $\mathbb{NP}$ complete
- The proof provides a theoretical upper bound $K_{theo} = 2|plan|(|C| + 1)$

domain	K_{theo}							
	min	max						
UMTranslog ●	70	1258						
Satellite ●	20	510						
SmartPhone ●	132	324						
Woodworking ●	12	48						
Monroe ●	198	11032						

Bounding Decomposition Height

Method 2:

- If every decomposition method would produce ≥ 2 tasks, then each decomposition increases the size of the plan

domain	K_{theo}							
	min	max						
UMTranslog ●	70	1258						
Satellite ●	20	510						
SmartPhone ●	132	324						
Woodworking ●	12	48						
Monroe ●	198	11032						

Bounding Decomposition Height

Method 2:

- If every decomposition method would produce $\geq$ 2 tasks, then each decomposition increases the size of the plan
- Methods where the task network contains only a single task are called *unit methods*

domain	K_{theo}							
	min	max						
UMTranslog ●	70	1258						
Satellite ●	20	510						
SmartPhone ●	132	324						
Woodworking ●	12	48						
Monroe ●	198	11032						

Bounding Decomposition Height

Method 2:

- If every decomposition method would produce $\geq$ 2 tasks, then each decomposition increases the size of the plan
- Methods where the task network contains only a single task are called *unit methods*
- Unit methods can be removed via expansion in the model

domain	K_{theo}							
	min	max						
UMTranslog ●	70	1258						
Satellite ●	20	510						
SmartPhone ●	132	324						
Woodworking ●	12	48						
Monroe ●	198	11032						

Bounding Decomposition Height

Method 2:

- If every decomposition method would produce ≥ 2 tasks, then each decomposition increases the size of the plan
- Methods where the task network contains only a single task are called *unit methods*
- Unit methods can be removed via expansion in the model
- Thus $K_{unit} = \frac{|plan|-1}{\delta-1}$

domain	K_{theo}							
	min	max						
UMTranslog	70	1258						
Satellite	20	510						
SmartPhone	132	324						
Woodworking	12	48						
Monroe	198	11032						

Bounding Decomposition Height

Method 2:

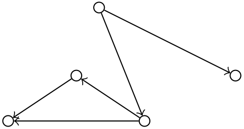
- If every decomposition method would produce ≥ 2 tasks, then each decomposition increases the size of the plan
- Methods where the task network contains only a single task are called *unit methods*
- Unit methods can be removed via expansion in the model
- Thus $K_{unit} = \frac{|plan|-1}{\delta-1}$

domain	K_{theo}		K_{unit}					
	min	max	min	max				
UMTranslog	70	1258	5	37				
Satellite	20	510	5	17				
SmartPhone	132	324	11	15				
Woodworking	12	48	2	4				
Monroe	198	11032	∞	∞				

Bounding Decomposition Height

Method 3:

- The TSTG describes how tasks can be decomposed into each other

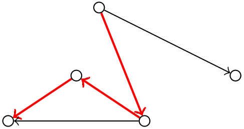


domain	K_{theo}		K_{unit}					
	min	max	min	max				
UMTranslog ●	70	1258	5	37				
Satellite ●	20	510	5	17				
SmartPhone ●	132	324	11	15				
Woodworking ●	12	48	2	4				
Monroe ●	198	11032	∞	∞				

Bounding Decomposition Height

Method 3:

- The TSTG describes how tasks can be decomposed into each other
- If acyclic, the longest path in the TSTG is an upper bound K_{TSTG}

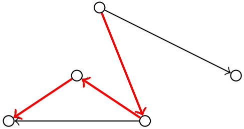


domain	K_{theo}		K_{unit}					
	min	max	min	max				
UMTranslog	70	1258	5	37				
Satellite	20	510	5	17				
SmartPhone	132	324	11	15				
Woodworking	12	48	2	4				
Monroe	198	11032	∞	∞				

Bounding Decomposition Height

Method 3:

- The TSTG describes how tasks can be decomposed into each other
- If acyclic, the longest path in the TSTG is an upper bound K_{TSTG}

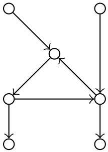


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog ●	70	1258	5	37	3	6		
Satellite ●	20	510	5	17	1	4		
SmartPhone ●	132	324	11	15	3	∞		
Woodworking ●	12	48	2	4	1	2		
Monroe ●	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic

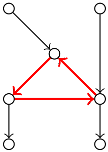


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog ●	70	1258	5	37	3	6		
Satellite ●	20	510	5	17	1	4		
SmartPhone ●	132	324	11	15	3	∞		
Woodworking ●	12	48	2	4	1	2		
Monroe ●	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods

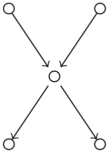


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog ●	70	1258	5	37	3	6		
Satellite ●	20	510	5	17	1	4		
SmartPhone ●	132	324	11	15	3	∞		
Woodworking ●	12	48	2	4	1	2		
Monroe ●	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods
- We can break these cycles by replacing them with a new abstract task

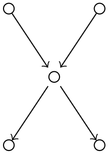


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog ●	70	1258	5	37	3	6		
Satellite ●	20	510	5	17	1	4		
SmartPhone ●	132	324	11	15	3	∞		
Woodworking ●	12	48	2	4	1	2		
Monroe ●	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods
- We can break these cycles by replacing them with a new abstract task
- Use a dynamic programming scheme to compute the $K_{t,n}$ necessary to capture all decompositions of a task t into n actions.

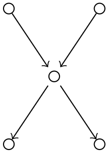


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog ●	70	1258	5	37	3	6		
Satellite ●	20	510	5	17	1	4		
SmartPhone ●	132	324	11	15	3	∞		
Woodworking ●	12	48	2	4	1	2		
Monroe ●	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods
- We can break these cycles by replacing them with a new abstract task
- Use a dynamic programming scheme to compute the $K_{t,n}$ necessary to capture all decompositions of a task t into n actions.
- $K_{DP} = K_{c_I,|plan|}$

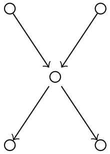


domain	K_{theo}		K_{unit}		K_{TSTG}			
	min	max	min	max	min	max		
UMTranslog	70	1258	5	37	3	6		
Satellite	20	510	5	17	1	4		
SmartPhone	132	324	11	15	3	∞		
Woodworking	12	48	2	4	1	2		
Monroe	198	11032	∞	∞	4	8		

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods
- We can break these cycles by replacing them with a new abstract task
- Use a dynamic programming scheme to compute the $K_{t,n}$ necessary to capture all decompositions of a task t into n actions.
- $K_{DP} = K_{c_I,|plan|}$

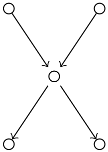


domain	K_{theo}		K_{unit}		K_{TSTG}		K_{DP}	
	min	max	min	max	min	max	min	max
UMTranslog ●	70	1258	5	37	3	6	3	6
Satellite ●	20	510	5	17	1	4	1	4
SmartPhone ●	132	324	11	15	3	∞	2	5
Woodworking ●	12	48	2	4	1	2	1	2
Monroe ●	198	11032	∞	∞	4	8	4	8

Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic, but only cycles of unit methods
- We can break these cycles by replacing them with a new abstract task
- Use a dynamic programming scheme to compute the $K_{t,n}$ necessary to capture all decompositions of a task t into n actions.
- $K_{DP} = K_{c_I,|plan|}$



domain	K_{theo}		K_{unit}		K_{TSTG}		K_{DP}	
	min	max	min	max	min	max	min	max
UMTranslog ●	70	1258	5	37	3	6	3	6
Satellite ●	20	510	5	17	1	4	1	4
SmartPhone ●	132	324	11	15	3	∞	2	5
Woodworking ●	12	48	2	4	1	2	1	2
Monroe ●	198	11032	∞	∞	4	8	4	8

$\Leftarrow$ take the minimum

Evaluation

- To ascertain the performance of our SAT-translation, we have conducted an empirical study on five common HTN benchmarking domains

domain	#instances	$\lvert L \rvert$		$\lvert C \rvert$		$\lvert A \rvert$		$\lvert M \rvert$	
		min	max	min	max	min	max	min	max
UMTranslog ●	21	19	88	4	16	7	22	4	17
Satellite ●	22	8	70	1	17	7	78	11	541
SmartPhone ●	3	44	47	5	8	16	18	14	99
Woodworking ●	5	32	59	1	4	6	24	4	76
Monroe ●	50	1220	3152	32	265	436	6017	408	5476

Evaluation

- To ascertain the performance of our SAT-translation, we have conducted an empirical study on five common HTN benchmarking domains

domain	#instances	$\|L\|$		$\|C\|$		$\|A\|$		$\|M\|$	
		min	max	min	max	min	max	min	max
UMTranslog ●	21	19	88	4	16	7	22	4	17
Satellite ●	22	8	70	1	17	7	78	11	541
SmartPhone ●	3	44	47	5	8	16	18	14	99
Woodworking ●	5	32	59	1	4	6	24	4	76
Monroe ●	50	1220	3152	32	265	436	6017	408	5476

domain	K_{theo}		K_{unit}		K_{TSTG}		K_{DP}	
	min	max	min	max	min	max	min	max
UMTranslog ●	70	1258	5	37	3	6	3	6
Satellite ●	20	510	5	17	1	4	1	4
SmartPhone ●	132	324	11	15	3	∞	2	5
Woodworking ●	12	48	2	4	1	2	1	2
Monroe ●	198	11032	∞	∞	4	8	4	8

- For 88 of 101 instances the computed height bound was exact.

Evaluation

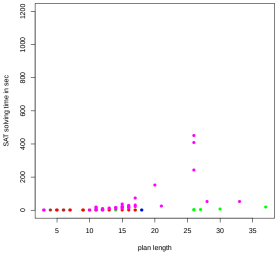


Figure: Runtime on actual solutions.

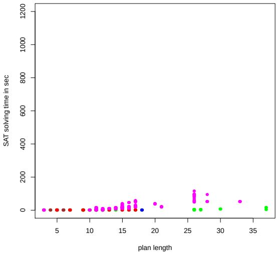


Figure: Runtime on non-solutions, generated by random-walking.

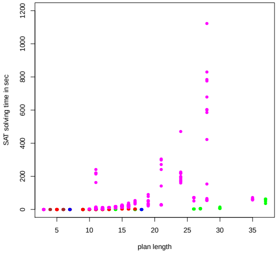


Figure: Runtime on non-solutions, generated by replacing a single action in a solution.

Conclusion

- We provided the first working plan verifier for HTN planning

Conclusion

- We provided the first working plan verifier for HTN planning
- ... and showed that plan verification possible in practice

Conclusion

- We provided the first working plan verifier for HTN planning
- ... and showed that plan verification possible in practice
- We showed that concise height bounds can be derived automatically from the domain

Conclusion

- We provided the first working plan verifier for HTN planning
- ... and showed that plan verification possible in practice
- We showed that concise height bounds can be derived automatically from the domain

- Promising directions of future research
  - Reducing the size of the encoding (still $\mathcal{O}(n^4)$)

Conclusion

- We provided the first working plan verifier for HTN planning
- ... and showed that plan verification possible in practice
- We showed that concise height bounds can be derived automatically from the domain

- Promising directions of future research
  - Reducing the size of the encoding (still $\mathcal{O}(n^4)$)
  - Creating a specialised formula for totally-ordered problems

Conclusion

- We provided the first working plan verifier for HTN planning
- ... and showed that plan verification possible in practice
- We showed that concise height bounds can be derived automatically from the domain

- Promising directions of future research
  - Reducing the size of the encoding (still $\mathcal{O}(n^4)$)
  - Creating a specialised formula for totally-ordered problems
  - Using the encoding as a SAT-based HTN planner