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 for totally ordered classical plans
- Validation (VAL) provides plan verification in classical domains
- NP-complete for HTN planning [Behnke et al. 2015]
- So far, no HTN plan verifier exists
Plan Verification

- $O(n)$ for totally ordered classical plans
Plan Verification

- $O(n)$ for totally ordered classical plans
- VAL provides for plan verification in classical domains
Plan Verification

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

- $O(n)$ 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

• 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
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?

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
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

\[ P = (P, C, c_i, M, L, s_i) \]

- \( P \) 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 \( tn \in \text{Sol}(P) \) 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 \( tn \in \text{Sol} (\mathcal{P}) \) 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

\[ P = (P, C, c_I, M, L, s_I) \]

- \( P \) 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 \( tn \in \text{Sol}(P) \) 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

\[ P = (P, C, c_I, M, L, s_I) \]
- \( P \) 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 \( tn \in \text{Sol}(\mathcal{P}) \) 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 \( t_n \in \text{Sol}(\mathcal{P}) \) 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
- \( M \subseteq C \times 2^\mathcal{TN} \) the methods
- \( L \) a set of variables
- \( s_I \subseteq L \) the initial state

A solution \( t_\mathcal{N} \in \text{Sol}(\mathcal{P}) \) 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 $tn \in Sol(\mathcal{P})$. 
Plan Verification

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

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 $tn \in \text{Sol}(\mathcal{P})$.

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 $tn \in Sol(\mathcal{P})$.

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 $tn \in \text{Sol}(\mathcal{P})$.

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 $tn \in Sol(\mathcal{P})$.

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 $tn \in Sol(\mathcal{P})$.

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

\[ l_n \]
Translation into SAT

Clauses describe local restrictions at position

- **Node Constraints**
  - at most one task

\[ l_n \circ \circ \circ \bullet \circ \circ \circ \circ \circ \circ \circ \]
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

![Diagram of hierarchical planning problem]

\[ I_{n-1} \]

\[ I_n \]
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

\[ l_{n-1} \]

\[ l_n \]

\[ l_{n+1} \]
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 \( \text{NP} \) complete
- The proof provides a theoretical upper bound

\[ K_{\text{theo}} = 2|\text{plan}|(|C| + 1) \]
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
  \[ K_{\text{theo}} = 2|\text{plan}|(|C| + 1) \]

<table>
<thead>
<tr>
<th>domain</th>
<th>( K_{\text{theo}} ) min</th>
<th>( K_{\text{theo}} ) max</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
</tr>
</tbody>
</table>
Bounding Decomposition Height

Method 2:

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

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td></td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td></td>
</tr>
</tbody>
</table>
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*.

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td></td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td></td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>domain</th>
<th>( K_{\text{theo}} )</th>
<th>( K_{\text{theo}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTranslog</td>
<td>min 70</td>
<td>max 1258</td>
</tr>
<tr>
<td>Satellite</td>
<td>min 20</td>
<td>max 510</td>
</tr>
<tr>
<td>Smartphone</td>
<td>min 132</td>
<td>max 324</td>
</tr>
<tr>
<td>Woodworking</td>
<td>min 12</td>
<td>max 48</td>
</tr>
<tr>
<td>Monroe</td>
<td>min 198</td>
<td>max 11032</td>
</tr>
</tbody>
</table>

\( \Rightarrow \) take the minimum
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.
- Thus $K_{unit} = \frac{|plan| - 1}{\delta - 1}$

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UMTTranslog</td>
<td>70</td>
<td>1258</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.
- Thus $K_{unit} = \frac{|plan| - 1}{\delta - 1}$

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
</tr>
</tbody>
</table>
Bounding Decomposition Height

Method 3:

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

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UMTranslog</td>
<td>70-1258</td>
<td>5-37</td>
</tr>
<tr>
<td>Satellite</td>
<td>20-510</td>
<td>5-17</td>
</tr>
<tr>
<td>Smartphone</td>
<td>132-324</td>
<td>11-15</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12-48</td>
<td>2-4</td>
</tr>
<tr>
<td>Monroe</td>
<td>198-11032</td>
<td>$\infty-\infty$</td>
</tr>
</tbody>
</table>
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 $K_{TSTG}$

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
</tr>
</tbody>
</table>
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 $K_{TSTG}$

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>∞</td>
</tr>
</tbody>
</table>
Bounding Decomposition Height

Method 4:

- Not all unit methods are problematic

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{\text{theo}}$</th>
<th>$K_{\text{unit}}$</th>
<th>$K_{\text{TSTG}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
Bounding Decomposition Height

Method 4:

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

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{\text{theo}}$</th>
<th>$K_{\text{unit}}$</th>
<th>$K_{\text{TSTG}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>∞</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTtranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>Smartphone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
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.
- $K_{DP} = K_{c_l,|plan|}$

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
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.

\[ K_{DP} = K_{c_i,|plan|} \]

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
<th>$K_{DP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>UMTTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>
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.

\[
K_{DP} = K_{c_l,|plan|}
\]

<table>
<thead>
<tr>
<th>domain</th>
<th>$K_{theo}$</th>
<th>$K_{unit}$</th>
<th>$K_{TSTG}$</th>
<th>$K_{DP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>UMTTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

$\Leftarrow$ 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 | $|L|_{\text{min}}$ | $|L|_{\text{max}}$ | $|C|_{\text{min}}$ | $|C|_{\text{max}}$ | $|A|_{\text{min}}$ | $|A|_{\text{max}}$ | $|M|_{\text{min}}$ | $|M|_{\text{max}}$ |
|-------------|------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| 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    |
| 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   |

<table>
<thead>
<tr>
<th>domain</th>
<th>(K_{theo})</th>
<th>(K_{unit})</th>
<th>(K_{TSTG})</th>
<th>(K_{DP})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>max</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td>UMTranslog</td>
<td>70</td>
<td>1258</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>510</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>SmartPhone</td>
<td>132</td>
<td>324</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Woodworking</td>
<td>12</td>
<td>48</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Monroe</td>
<td>198</td>
<td>11032</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
</tbody>
</table>

- 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 $O(n^4)$)
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 $O(n^4)$)
- 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