Tree-REX: SAT-based Tree Exploration
For Efficient and High-Quality HTN Planning

Dominik Schreiber, Damien Pellier, Humbert Fiorino, Tomáš Balyo | July 13, 2019
Outline

- Introduction
  - {Automated, Hierarchical, SAT-based} Planning
- Related Work
- Tree-REX Planning approach
  - Algorithm; Encoding; Plan length optimization
- Evaluation
- Conclusion
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- (World) **State**: Consistent set of boolean atoms; e.g. \( \text{at(ball,A), at(robot,B)} \)
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- **Action a**: Has boolean preconditions and effects; e.g. action `move(robot,A,B)` requires `at(robot,A)`, deletes `at(robot,A)`, adds `at(robot,B)`
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- **Goal** $g$: Subset of possible states, e.g. $\text{at(ball},B)$ must hold
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  
  e.g. $\pi =$
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- Plan $\pi$: Action sequence transforming an initial state to a goal state
e.g. $\pi = \langle\text{pickup(robot,ball,A)}\rangle$
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  e.g. $\pi = \langle \text{pickup(robot,ball,A)}, \text{move(robot,A,B)} \rangle$
Automated Planning

Find a valid sequence of actions from some initial world state to a desired goal state.

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  
e.g. $\pi = \langle \text{pickup(robot, ball, A)}, \text{move(robot, A, B)}, \text{drop(robot, ball, B)} \rangle$
Hierarchical Planning

Main idea: Share *domain-specific expert knowledge* with your planner.

- Which tasks need to be achieved
- How to directly achieve simple tasks
- How to break down complex tasks into simpler ones
Hierarchical Planning

Main idea: *Share domain-specific expert knowledge with your planner.*
- Which **tasks** need to be achieved
- How to **directly achieve simple tasks**
- How to **break down complex tasks** into simpler ones

Most popular: **Hierarchical Task Network (HTN) Planning** [Erol et al., 1994]
- Extension of classical planning (states, actions, plans)
- More expressive than classical planning
- More focused search, enables more efficient planning
Hierarchical Task Networks

move-ball(ball,B)

navigate(robot,B,A)
pickup(robot,ball,A)
navigate(robot,A,B)
drop(robot,ball,B)

move(robot,B,A)
move(robot,A,B)
Hierarchical Task Networks

Root(s): Initial task(s), part of problem input

- Abstract notion of what needs to be achieved
Hierarchical Task Networks

Directed edges: Subtask relationships
- Totally ordered HTN planning \(\Rightarrow\) Total order on subtasks
- Span a tree of tasks
Hierarchical Task Networks

Inner nodes: **Compound tasks**

- Can be achieved by picking a **method** and achieving each subtask
- Example: Method $\text{move-ball}(\text{ball, to, r, x, y})$
  
  **Preconditions** \{ at(ball,x), at(r,y) \},

  **Subtasks** (1) $\text{navigate}(r,y,x)$, (2) $\text{pickup}(r,\text{ball,x})$,
  (3) $\text{navigate}(r,x,\text{to})$, (4) $\text{drop}(r,\text{ball,to})$
Hierarchical Task Networks

Leaf nodes: **Primitive tasks**

- Directly correspond to applying a certain action
- In-order traversal of all leaves ⇒ **Plan!**
SAT-based (Classical) Planning

Iteratively encode planning problem into propositional logic up to a certain number $n$ of steps [Kautz and Selman, 1992]
SAT-based Totally Ordered HTN Planning


- Encodings do not address recursive task relationships
  (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps
SAT-based Totally Ordered HTN Planning


- Encodings do not address recursive task relationships (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps

2018: totSAT [Behnke et al., 2018]

- Encode problem’s hierarchy up to depth \( k \) until satisfiable for some \( k \)
- Method preconditions compiled into “virtual actions”
SAT-based Totally Ordered HTN Planning

- Encodings do not address recursive task relationships  
  (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps

2018: totSAT  [Behnke et al., 2018]
- Encode problem’s hierarchy up to depth $k$ until satisfiable for some $k$
- Method preconditions compiled into “virtual actions”

2019: SAT-based Stack Machine Simulation  [Schreiber et al., 2019]
- Uses incremental SAT solving: Maintains a single logical formula
- Requires hyperparameter (maximum stack size to encode)
Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold
Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold

Layer 0:
3 initial tasks
Tree-like Reduction Exploration

Explore hierarchy \textit{breadth-first (layer by layer)}

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold

Layer 0:
3 initial tasks

Layer 1
Tree-like Reduction Exploration

Explore hierarchy **breadth-first (layer by layer)**

- Each layer contains **potential abstract plans**
- Expand hierarchy until actual valid plan is found
  - **Primitive subtasks only**
  - All preconditions, effects, goals hold

Layer 0:
3 initial tasks

Layer 1

Layer 2
Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)
- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold

Layer 0: 3 initial tasks
Layer 1
Layer 2: Plan!
Algorithm: Tree-REX Planning Procedure

1. \( \Pi := \) Preprocess and ground HTN planning problem;
2. \( C := \emptyset; \) // Logical clauses
3. \( \text{nextLayer} := \) GenerateFirstLayer(\( \Pi \));
4. \( \text{prevLayer} := \text{nextLayer}; \)
5. \( \text{result} := \) NONE;
6. while \( \text{result} = \) NONE do
   7. \( \text{prevLayer} := \text{nextLayer}; \)
   8. \( \text{nextLayer} := \) Expand(\( \text{nextLayer} \)); // Create new hierarchical layer
   9. \( C := C \cup \) Encode(\( \text{prevLayer}, \text{nextLayer} \)); // Add clauses
10. \( \text{result} := \) Solve(\( C \), AssumeAllPrimitive(\( \text{nextLayer} \))); // SAT Solving
11. return \( \) Decode(\( \Pi \), \( \text{result} \)). // Extract plan from sat. assignment
Tree-REX: Algorithm

Algorithm: Tree-REX Planning Procedure

1: $\Pi := \text{Preprocess and ground HTN planning problem};$
2: $C := \emptyset;$   \hspace{1cm} // Logical clauses
3: $nextLayer := \text{GenerateFirstLayer}(\Pi);$  
4: $prevLayer := nextLayer;$  
5: $result := \text{NONE};$
6: while $result = \text{NONE}$ do 
7: $prevLayer := nextLayer;$  
8: $nextLayer := \text{Expand}(nextLayer);$   \hspace{1cm} // Create new hierarchical layer
Algorithm: Tree-REX Planning Procedure

1: $\Pi := \text{Preprocess and ground HTN planning problem}$;
2: $C := \emptyset$; \hspace{1cm} // Logical clauses
3: $\text{nextLayer} := \text{GenerateFirstLayer}(\Pi)$;
4: $\text{prevLayer} := \text{nextLayer}$;
5: $\text{result} := \text{NONE}$;
6: \textbf{while} $\text{result} = \text{NONE}$ \textbf{do}
7: \hspace{1cm} $\text{prevLayer} := \text{nextLayer}$;
8: \hspace{1cm} $\text{nextLayer} := \text{Expand}(\text{nextLayer})$; \hspace{1cm} // Create new hierarchical layer
9: \hspace{1cm} $C := C \cup \text{Encode}(\text{prevLayer}, \text{nextLayer})$; \hspace{1cm} // Add clauses
10: \hspace{1cm} $\text{result} := \text{Solve}(C, \text{AssumeAllPrimitive}(\text{nextLayer}))$; \hspace{1cm} // SAT Solving
11: \textbf{return} $\text{Decode}(\Pi, \text{result})$. \hspace{1cm} // Extract plan from sat. assignment
Tree-REX: Algorithm

Algorithm: Tree-REX Planning Procedure

1: \( \Pi := \text{Preprocess and ground HTN planning problem;} \)
2: \( C := \emptyset; \) \hspace{1cm} // Logical clauses
3: \( \text{nextLayer} := \text{GenerateFirstLayer}(\Pi); \)
4: \( \text{prevLayer} := \text{nextLayer}; \)
5: \( \text{result} := \text{NONE}; \)
6: \( \text{while} \ \text{result} = \text{NONE do} \)
7: \( \text{prevLayer} := \text{nextLayer}; \)
8: \( \text{nextLayer} := \text{Expand}(\text{nextLayer}); \) \hspace{1cm} // Create new hierarchical layer
9: \( C := C \cup \text{Encode}(\text{prevLayer}, \text{nextLayer}); \) \hspace{1cm} // Add clauses
10: \( \text{result} := \text{Solve}(C, \text{AssumeAllPrimitive}(\text{nextLayer})); \) \hspace{1cm} // SAT Solving
11: \( \text{return} \ \text{Decode}(\Pi, \text{result}). \) \hspace{1cm} // Extract plan from sat. assignment
Tree-REX: Encoding into SAT

Boolean variables:

- Variable for each possible fact/action/method at each position \((l, i)\)
- Variable indicating primitiveness of each position \((l, i)\)
Tree-REX: Encoding into SAT

Boolean variables:
- Variable for each possible fact/action/method at each position \((l, i)\)
- Variable indicating primitiveness of each position \((l, i)\)

Clauses:
- Initial layer: Enforce initial task network
- Every layer: Classical planning clauses for occurring facts, actions [Kautz and Selman, 1992]
- In between layers: Propagation of facts, actions, Reduction of methods into any of its possible children
- Final layer: Enforce actions only (no methods!)
Solution is found at layer $m$: Plan has some length $n \leq \text{size}(\text{lastLayer})$

$\Rightarrow$ Length of plan $\equiv$ Amount of actual actions at final layer

$\Rightarrow$ No incentive for SAT solver to prefer solutions with fewer actions
Tree-REX: Plan Length Optimization

Solution is found at layer \( m \): Plan has some length \( n \leq \text{size(lastLayer)} \)

\[ \Rightarrow \] Length of plan \( \equiv \) Amount of actual actions at final layer

\[ \Rightarrow \] No incentive for SAT solver to prefer solutions with fewer actions

**Quality awareness**: Use incremental SAT solving to find better plans

- Encode simple plan length counter at final layer:
  - Boolean variable for “plan length is greater than or equal to \( x \)”
Tree-REX: Plan Length Optimization

Solution is found at layer $m$: Plan has some length $n \leq \text{size(lastLayer)}$

$\Rightarrow$ Length of plan $\equiv$ Amount of actual actions at final layer

$\Rightarrow$ No incentive for SAT solver to prefer solutions with fewer actions

Quality awareness: Use incremental SAT solving to find better plans

- Encode simple plan length counter at final layer: Boolean variable for “plan length is greater than or equal to $x$”
- Enforce plan length smaller than $n$ as an additional assumption
  $\Rightarrow$ SAT: Better plan has been found (length $< n$)
  $\Rightarrow$ UNSAT: Plan length $n$ is already optimal at this layer

Anytime procedure: Better results the more resources are invested
Evaluations

- PDDL benchmarks from previous HTN planning evaluations
  [Behnke et al., 2018, Ramoul et al., 2017]
- Competitors:
  - Tree-REX (no plan length optimization)
  - Tree-REX-o (with plan length optimization)
  - totSAT  [Behnke et al., 2018] (AAAI-18 configuration)
- SAT solver: MiniSat  [Eén and Sörensson, 2003] for all competitors
- Up to 5 minutes of total run time
<table>
<thead>
<tr>
<th>Domain</th>
<th>#Inst.</th>
<th>totSAT</th>
<th>Tree-REX</th>
<th>Tree-REX-o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman</td>
<td>20</td>
<td>5.00</td>
<td>20.00</td>
<td>9.19</td>
</tr>
<tr>
<td>Blocksworld</td>
<td>20</td>
<td>4.02</td>
<td>20.00</td>
<td>19.22</td>
</tr>
<tr>
<td>Childsnack</td>
<td>20</td>
<td>0.91</td>
<td>20.00</td>
<td>19.99</td>
</tr>
<tr>
<td>Depots</td>
<td>20</td>
<td>7.81</td>
<td>19.36</td>
<td>18.95</td>
</tr>
<tr>
<td>Entertainment</td>
<td>12</td>
<td>6.73</td>
<td>9.25</td>
<td>8.94</td>
</tr>
<tr>
<td>Gripper</td>
<td>20</td>
<td>2.92</td>
<td>20.00</td>
<td>19.85</td>
</tr>
<tr>
<td>Hiking</td>
<td>20</td>
<td>0.93</td>
<td>20.00</td>
<td>17.93</td>
</tr>
<tr>
<td>Rover</td>
<td>20</td>
<td>0.87</td>
<td>20.00</td>
<td>10.12</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>1.23</td>
<td>16.00</td>
<td>8.62</td>
</tr>
<tr>
<td>Transport</td>
<td>30</td>
<td>3.77</td>
<td>30.00</td>
<td>21.51</td>
</tr>
</tbody>
</table>

Table: Run time IPC scores
## Evaluations: Plan Quality

<table>
<thead>
<tr>
<th>Domain</th>
<th>#Inst.</th>
<th>totSAT</th>
<th>Tree-REX</th>
<th>Tree-REX-o</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman</td>
<td>20</td>
<td>19.67</td>
<td>18.67</td>
<td>20.00</td>
</tr>
<tr>
<td>Blocksworld</td>
<td>20</td>
<td>18.36</td>
<td>19.68</td>
<td>19.68</td>
</tr>
<tr>
<td>Childsnack</td>
<td>20</td>
<td>12.00</td>
<td>20.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Depots</td>
<td>20</td>
<td>18.59</td>
<td>19.74</td>
<td>19.93</td>
</tr>
<tr>
<td>Entertainment</td>
<td>12</td>
<td>11.92</td>
<td>11.79</td>
<td>12.00</td>
</tr>
<tr>
<td>Gripper</td>
<td>20</td>
<td>20.00</td>
<td>20.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Hiking</td>
<td>20</td>
<td>11.00</td>
<td>20.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Rover</td>
<td>20</td>
<td>4.83</td>
<td>13.08</td>
<td>20.00</td>
</tr>
<tr>
<td>Satellite</td>
<td>20</td>
<td>9.80</td>
<td>9.66</td>
<td>16.00</td>
</tr>
<tr>
<td>Transport</td>
<td>30</td>
<td>23.71</td>
<td>26.04</td>
<td>30.00</td>
</tr>
</tbody>
</table>

Table: Plan length IPC scores
Conclusion

- Tree-REX: SAT-based totally ordered HTN planning
- Rapid encoding and solving through incremental SAT solving
- Anytime plan length optimization
  ⇒ First quality-aware SAT-based HTN planner
- Outperforms state-of-the-art SAT-based planning regarding run times and/or plan quality
Conclusion

- Tree-REX: SAT-based totally ordered HTN planning
- Rapid encoding and solving through incremental SAT solving
- Anytime plan length optimization
  ⇒ First quality-aware SAT-based HTN planner
- Outperforms state-of-the-art SAT-based planning regarding run times and/or plan quality

Future work

- Integrate into framework with additional features
  (disjunctive / ADL conditions, conditional effects, . . .)
- Interleave expansion and resolution of hierarchical layers
- Investigate (partially) lifted SAT encodings for HTN
References I

TotSAT—totally-ordered hierarchical planning through SAT.

Temporal induction by incremental SAT solving.

UMCP: A sound and complete procedure for hierarchical task-network planning.

Planning as Satisfiability.
Encoding HTN planning in propositional logic.
In *Proceedings International Conference on Artificial Intelligence Planning and Scheduling*, pages 190–198.

Grounding of HTN planning domain.

Efficient SAT encodings for hierarchical planning.
Hierarchical Planning

- Generally, can allow partially ordered and interleaving subtasks (here: total order on subtasks)
- Difficulty of HTN planning: Choosing the “right” method to fulfill a task (non-deterministic / randomized choice, heuristics, . . . )

Compared to classical planning:
- Smaller, more directed search space to find a plan
- More expressive [Erol et al., 1994]
Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods

- **Lookup** for which objects need to be encoded where at layer $l$
- $(l, i) := i$-th position of layer $l$
- $x \in (l, i) :\iff$ encode method / action / fact $x$ at position $(l, i)$
Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods
- Lookup for which objects need to be encoded where at layer $l$
- $(l, i) := i$-th position of layer $l$
- $x \in (l, i) :\iff$ encode method / action / fact $x$ at position $(l, i)$

Inductive construction of layers:
- Layer 0: Fully defined by initial state and initial task network
- Layer $n - 1 \leadsto n$: Propagate facts and actions, reduce methods, add facts implied by new actions and methods
Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods

- **Lookup** for which objects need to be encoded where at layer $l$
- $(l, i) := i$-th position of layer $l$
- $x \in (l, i) \iff$ encode method / action / fact $x$ at position $(l, i)$

Inductive construction of layers:

- Layer 0: Fully defined by initial state and initial task network
- Layer $n - 1 \mapsto n$: Propagate facts and actions, reduce methods, add facts implied by new actions and methods

Each subtask may have many different realizations to choose from!

- Example: $\text{walk}(a, b)$ may be achieved by walking over $c$, over $d$, ...
- Encode all options – SAT solver will decide on which option to take
Example: Rover planning problem with one initial task, showing only one of the possible actions/methods at each position.

Layer 0:
- get_soil_data(w0)
- navigate(w0)
- empty_store()

Layer 1:
- send_soil_data(w0)
- sample_soil(w0)

Layer 2:
- do_navigate(w1,w0)
  - communicate(w0,w1)
  - sample_soil(w0)
  - nop()
  - unvisit(w1)
  - visit(w1)

Layer 3:
- move(w1,w0)
  - communicate(w0,w1)
  - sample_soil(w0)
  - nop()
  - unvisit(w1)
  - visit(w1)
Tree-like Reduction Exploration (3/2)

**Example**: Rover planning problem with one initial task, showing only one of the possible actions/methods at each position.

Layer 3: Valid sequence of primitive actions $\Rightarrow$ Finished plan!
Tree-REX: Encoding into SAT (1/2)

Boolean variables:

- Variable for each possible fact/action/method at each position $(l, i)$
- Variable indicating primitiveness of each position $(l, i)$
Tree-REX: Encoding into SAT (1/2)

Boolean variables:
- Variable for each possible fact/action/method at each position \((l, i)\)
- Variable indicating primitiveness of each position \((l, i)\)

Classical planning clauses: mostly from [Kautz and Selman, 1992]

1. The initial state holds at \((l, 0)\) and the goals hold at \((l, s_l)\).
2. An action at \((l, i)\) implies its preconditions at \((l, i)\) and its effects at \((l, i + 1)\).
3. At most one action occurs at each position \((l, i)\).
4. A fact changes its value from \((l, i)\) to \((l, i + 1)\) only if some action supporting the fact change occurs at \((l, i)\) or \((l, i)\) is not primitive.
Tree-REX: Encoding into SAT (2/2)

Primitiveness and method preconditions:

5. An action (method) at position $p$ implies that $p$ is (not) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$. 
Primitiveness and method preconditions:

5. An action (method) at position $p$ implies that $p$ is (not) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$.

Transitions from layer $l$ to layer $l + 1$, former position $i$ shifted to $i' \geq i$:

7. A fact holds at $(l, i)$ iff it holds at $(l + 1, i')$.
8. An action at $(l, i)$ implies the same action at $(l + 1, i')$.
9. A method at $(l, i)$ implies any of its valid children at $(l + 1, i'), \ldots, (l + 1, i' + k)$.
Primitiveness and method preconditions:
5. An action (method) at position $p$ implies that $p$ is (not) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$.

Transitions from layer $l$ to layer $l + 1$, former position $i$ shifted to $i' \geq i$:
7. A fact holds at $(l, i)$ iff it holds at $(l + 1, i')$.
8. An action at $(l, i)$ implies the same action at $(l + 1, i')$.
9. A method at $(l, i)$ implies any of its valid children at $(l + 1, i'), \ldots, (l + 1, i' + k)$.

Initial and final layer:
10. Each initial task $t_i$ implies some according method/action at $(0, i)$.
11. Each position of the final layer is primitive. (Goal assumptions)
Tree-REX: Realization

Encoding

- Enforce virtual “blank” actions where positions should remain empty
- Additional (redundant) clauses: Backwards propagation – Necessary conditions for an action / method to occur somewhere
- Various optimizations reducing clauses and/or variables
Tree-REX: Realization

Encoding
- Enforce virtual “blank” actions where positions should remain empty
- Additional (redundant) clauses: Backwards propagation – Necessary conditions for an action / method to occur somewhere
- Various optimizations reducing clauses and/or variables

Implementation
- Preprocessing and grounding from [Ramoul et al., 2017]
- Separate interpreter application
  – Receives abstract encoding “blueprint”
  – Instantiates clauses as necessary
- IPASIR as generic incremental SAT interface
  ⇒ Plug in any modern incremental SAT solver
Expansion of Methods.

Consider the $l$-th and $(l + 1)$-th hierarchical layers of an HTN planning problem. For some $i \geq 0$, all facts and actions at $(l, i)$ are propagated to $(l + 1, i')$, where $i' \geq i$. Let method $r$ occur at $(l, i)$.

- If the $k$-th subtask $t_k$ of $r$ is primitive, the corresponding action $a_{t_k}$ is added to $(l + 1, i')$.
- If $t_k$ is compound, each possible corresponding method $r' \in R(t_k)$ is added to $(l + 1, i' + k)$.

How to incrementally build formula layer by layer?

- Add clauses (9) once, clauses (1-8) for each new layer $l$
- Assume clauses (10) before each solving attempt (drop afterwards)
Evaluation Plots I

Schreiber et al. – SAT-based Tree-Exploration for HTN Planning
July 13, 2019
Evaluations: Insights

- Tree-REX dominates run times, Tree-REX-o dominates plan quality
- Plan length optimization heavily domain-dependent
  - Not improvable due to rigid hierarchy (e.g. Childsnack, Gripper): instant termination after initial plan is found
  - Slightly improvable (e.g. Barman): quick optimization process
  - Highly improvable (e.g. Rover, Satellite): long optimization process with many little improvements
- Bottleneck for large instances: Grounding, SAT solving