Efficient SAT Encodings for Hierarchical Planning

11th International Conference on Agents and Artificial Intelligence
Dominik Schreiber (Speaker), Tomáš Balyo, Damien Pellier, Humbert Fiorino | February 19, 2019

```
(tag t3 (serve_shot ?sh ?c))
  :constraints(and
    (before (and (ontable ?sh) (clean ?sh) (handempty ?h)) t1)
  )
)

(:method do_cocktail_in_shot2
  :parameters (?sh - shot ?i - ingredient)
  :expansion (sum
    (tag t1 (do_fill_shot ?sh ?i ?h))
    (tag t2 (leave ?h ?sh))
  )
  :constraints(and
    (before (and (dispenses ?d ?i)) t1)
  )
)
```
Outline

- Background: Planning, Hierarchical Planning, SAT Planning
- Related Work
- Contributions: GCT Encoding, SMS Encoding
- 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. \(\text{at(ball,A)}, \text{at(robot,B)}\)
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 $\text{move}(\text{robot}, A, B)$ requires $\text{at}(\text{robot}, A)$, deletes $\text{at}(\text{robot}, A)$, adds $\text{at}(\text{robot}, B)$
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. $\pi = \langle$ pickup(robot, ball, A), move(robot, A, B) $\rangle$
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. $\pi = \langle \text{pickup(robot,ball,A), move(robot,A,B), drop(robot,ball,B)} \rangle$
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 (same 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

- Here: totally ordered
- Span a tree of tasks
Hierarchical Task Networks

Inner nodes: Compound tasks

- Can be achieved by choosing a method and achieving each subtask
- Example: method move-ball(ball, to, r, x, y)
  
  Preconditions \{ at(ball,x), at(r,y) \},
  
  Subtasks \langle (1) navigate(r,y,x), (2) pickup(r,ball,x),
  (3) navigate(r,x,to), (4) drop(r,ball,to) \rangle
Hierarchical Task Networks

Leaf nodes: **Primitive tasks**

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

- Encode planning problem into propositional logic up to a certain number of steps [Kautz and Selman, 1992]
- Use Satisfiability (SAT) Solver to find satisfying assignment
- Decode assignment back into a plan

Schreiber et al. – Efficient SAT Encodings for Hierarchical Planning

February 19, 2019
Related Work

Introduction of **incremental SAT solving** to planning

[Gocht and Balyo, 2017]

- Maintain and iteratively extend a **single logical formula**
- Remember logical conflicts from previous iterations
Related Work

Introduction of **incremental SAT solving** to planning
[Gocht and Balyo, 2017]

- Maintain and iteratively extend a **single logical formula**
- Remember logical conflicts from previous iterations

SAT-based HTN planning: Few research before 2018
[Mali and Kambhampati, 1998]

- Previous 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**
- In practice, infeasible for today’s problem instances
Enhancement of previous *bottom-up linear forward* encoding

[Mali and Kambhampati, 1998]

- Focused on **totally ordered** HTN planning
- Fully supporting **recursive subtask relationships**
- Resulting in **smaller encoding size** (quadratic in #steps, #tasks)
Encoding: Grammar-Constrained Tasks

Enhancement of previous *bottom-up linear forward* encoding [Mali and Kambhampati, 1998]

- Focused on *totally ordered* HTN planning
- Fully supporting *recursive subtask relationships*
- Resulting in *smaller encoding size* (quadratic in #steps, #tasks)

Limitations of new encoding:

- Encoding *still too large* for realistic problem sizes
- Allows for *interleaving of tasks* in some special cases
Encoding: Grammar-Constrained Tasks

Enhancement of previous *bottom-up linear forward* encoding
[Mali and Kambhampati, 1998]

- Focused on *totally ordered* HTN planning
- Fully supporting *recursive subtask relationships*
- Resulting in *smaller encoding size* (quadratic in #steps, #tasks)

Limitations of new encoding:

- Encoding *still too large* for realistic problem sizes
- Allows for *interleaving of tasks* in some special cases

Observation: HTN is like enforcing a grammar on valid plans

- Totally ordered HTN corresponds to *context-free grammar*
- Finding a plan equivalent to *deriving a word from the grammar*
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add **transition rules** *(pop, push)* to process tasks until stack is empty

Plan: ⟨
  move-ball(ball,B)
⟩
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add **transition rules** (`pop`, `push`) to process tasks until stack is empty

Plan: ⟨⟩
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add **transition rules** (pop, push) to process tasks until stack is empty

Plan: ⟨

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

```
move(robot, B,A)  
navigate(robot, A,B)  
pickup(robot, ball,A)  
```

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

```drop(robot, ball,B)  
navigate(robot, A,B)  
pickup(robot, ball,A)  
move(robot, B,A)
```
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

```
move(robot, B,A)
navigate(robot, A,B)
pickup(robot, ball,A)
drop(robot, ball,B)
navigate(robot, A,B)
pickup(robot, ball,A)
drop(robot, ball,B)
→
moved
```
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add transition rules (**pop**, **push**) to **process tasks until stack is empty**

Plan: \( \langle \text{move(robot,room2,room1)}, \text{pickup(robot,ball,room1)} \rangle \)
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add **transition rules** (*pop*, *push*) to **process tasks** until stack is empty

Plan: \( \langle \text{move(\text{robot,room2,room1})}, \text{pickup(\text{robot,ball,room1})} \rangle \)
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:
- Encode **stack of tasks** at each step of future plan
- Add **transition rules** (pop, push) to process tasks until stack is empty

Plan: \( \langle \text{move(robot, room2, room1)}, \text{pickup(robot, ball, room1)}, \text{move(robot, room1, room2)} \rangle \)
Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode **stack of tasks** at each step of future plan
- Add **transition rules** (*pop*, *push*) to process tasks until stack is empty

Plan: $\langle$ move(robot,room2,room1), pickup(robot,ball,room1),
move(robot,room1,room2), drop(robot,ball,room2) $\rangle$
Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step

Furthermore, all clauses only contain variables from adjacent steps, which allows the formula to be expanded incrementally.

Assertion to SAT solver: stack must be empty at final step.

⇒ Assignment found: Extract plan from true action variables

⇒ Unsatisfiable: Increase $n$, add new clauses, repeat

Properties:
- Handles all special cases (recursive subtasks, no interleaving, etc.)
- Requires parameter $\sigma$: Maximum stack size to encode

$O(\#\text{steps} \cdot (\sigma \cdot \#\text{tasks} + \#\text{methods} + \#\text{actions}))$ clauses
Encoding: Stack Machine Simulation (2)

Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  ⇒ Formula can be expanded incrementally
Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  ⇒ Formula can be expanded incrementally
- Assertion to SAT solver: stack must be empty at final step $n$
  ⇒ Assignment found: Extract plan from true action variables
  ⇒ Unsatisfiable: Increase $n$, add new clauses, repeat
Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  ⇒ Formula can be expanded incrementally
- Assertion to SAT solver: stack must be empty at final step \( n \)
  ⇒ Assignment found: Extract plan from true action variables
  ⇒ Unsatisfiable: Increase \( n \), add new clauses, repeat

Properties

- Handles all special cases (recursive subtasks, no interleaving, etc.)
- Requires parameter \( \sigma \): Maximum stack size to encode
- \( \mathcal{O}(\#steps \cdot (\sigma \cdot \#tasks + \#methods + \#actions)) \) clauses
Internal evaluation of approaches

- GCT Encoding
- SMS Encoding (3 variants)
Evaluation

Internal evaluation of approaches

- GCT Encoding
- SMS Encoding (3 variants)

Evaluation environment:

- 120 benchmark instances from six IPC domains
  - Barman, Blocksworld, Childsnack, Elevator, Rover, Satellite
- 24 core Intel Xeon CPU E5-2630 @ 2.30 GHz, 264 GB of RAM
- Limits per run: five minutes; 12 GB of RAM
Comparison of Run Times

![Graph showing the comparison of run times for GCT, SMS-bt, SMS-ud, and SMS-ut. The x-axis represents the number of solved instances, and the y-axis represents time limit in seconds. The graph shows the performance of each method across various time limits and numbers of solved instances.]
## Run Time Scores per Domain

<table>
<thead>
<tr>
<th>Domain</th>
<th>GCT</th>
<th>SMS-bt</th>
<th>SMS-ut</th>
<th>SMS-ur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman</td>
<td>0.09</td>
<td>1.90</td>
<td>1.96</td>
<td>4.68</td>
</tr>
<tr>
<td>Blocksworld</td>
<td>0.08</td>
<td>9.22</td>
<td>10.94</td>
<td>6.74</td>
</tr>
<tr>
<td>Childsnack</td>
<td>0.98</td>
<td>3.90</td>
<td>9.95</td>
<td>4.50</td>
</tr>
<tr>
<td>Elevator</td>
<td>4.21</td>
<td>14.86</td>
<td>13.32</td>
<td>10.29</td>
</tr>
<tr>
<td>Rover</td>
<td>0.44</td>
<td>6.17</td>
<td>5.40</td>
<td>5.58</td>
</tr>
<tr>
<td>Satellite</td>
<td>0.96</td>
<td>7.08</td>
<td>7.17</td>
<td>16.08</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>6.75</td>
<td>43.13</td>
<td>48.74</td>
<td>47.88</td>
</tr>
</tbody>
</table>

Score for each instance and competitor: \( \frac{T^*}{T} = \frac{\text{best competitor's run time}}{\text{run time}} \)
## Plan Length Scores per Domain

<table>
<thead>
<tr>
<th>Domain</th>
<th>GCT</th>
<th>SMS-bt</th>
<th>SMS-ut</th>
<th>SMS-ur</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barman</td>
<td>0.85</td>
<td>2.72</td>
<td>2.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Blocksworld</td>
<td>2.00</td>
<td>10.00</td>
<td>13.00</td>
<td>11.00</td>
</tr>
<tr>
<td>Childsnack</td>
<td>3.00</td>
<td>6.00</td>
<td>10.00</td>
<td>8.00</td>
</tr>
<tr>
<td>Elevator</td>
<td>13.00</td>
<td>16.00</td>
<td>15.00</td>
<td>15.00</td>
</tr>
<tr>
<td>Rover</td>
<td>3.86</td>
<td>6.62</td>
<td>6.55</td>
<td>6.62</td>
</tr>
<tr>
<td>Satellite</td>
<td>4.00</td>
<td>9.61</td>
<td>11.79</td>
<td>16.77</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>26.70</td>
<td>50.96</td>
<td>58.33</td>
<td>62.40</td>
</tr>
</tbody>
</table>

Score for each instance and competitor: \( \frac{T^*}{T} = \frac{\text{best competitor’s plan length}}{\text{plan length}} \)
Two new SAT encodings for totally ordered HTN planning

GCT: Handles recursive subtask relationships
SMS: Introduces incremental SAT solving to HTN planning
Conclusion

- Two new SAT encodings for totally ordered HTN planning
  - GCT: Handles recursive subtask relationships
  - SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Future work

- Enhance SMS to expand tasks more rapidly
- Eliminate hyper-parameter $\sigma$ by changing structure of encoding
- Compare to recent related work [Behnke et al., 2018]

Thank you for your attention!
Conclusion

- Two new SAT encodings for totally ordered HTN planning
  - GCT: Handles recursive subtask relationships
  - SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Future work
- Enhance SMS to expand tasks more rapidly
- Eliminate hyper-parameter $\sigma$ by changing structure of encoding
- Compare to recent related work [Behnke et al., 2018]
Conclusion

- Two new SAT encodings for totally ordered HTN planning
  - GCT: Handles recursive subtask relationships
  - SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Future work

- Enhance SMS to expand tasks more rapidly
- Eliminate hyper-parameter $\sigma$ by changing structure of encoding
- Compare to recent related work [Behnke et al., 2018]

Thank you for your attention!
References I

totSAT–totally-ordered hierarchical planning through SAT.
In *Proceedings of the 32th AAAI conference on AI (AAAI 2018).* AAAI Press.

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

Accelerating SAT based planning with incremental SAT solving.
*Proceedings of the International Conference on Automated Planning and Scheduling*, pages 135–139.

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