Motivation

Planning systems require some planning knowledge

- Simple STRIPS planners need only operators, which describe how actions change the world
- More advanced planners require operators and additional planning knowledge for problem solving

Problem

- Developing additional planning knowledge for complex planners is a difficult manual process performed by domain experts
Planning systems require some planning knowledge

- Simple STRIPS planners need only operators, which describe how actions change the world
- More advanced planners require operators and additional planning knowledge for problem solving

Problem

- Developing additional planning knowledge for complex planners is a difficult manual process performed by domain experts
Planning systems require some planning knowledge

- Simple STRIPS planners need only operators, which describe how actions change the world
- More advanced planners require operators and additional planning knowledge for problem solving

Problem

- Developing additional planning knowledge for complex planners is a difficult manual process performed by domain experts
Objective

- We describe an algorithm for learning such planning knowledge for a certain class of advanced planners known as Hierarchical Task Network (HTN) planners.
  - Actually, a specific form of HTN planning known as Ordered Task Decomposition (OTD).
  - OTD planners are useful because of potential for efficiency and expressivity; used in most fielded applications.
Objective

- We describe an algorithm for learning such planning knowledge for a certain class of advanced planners known as Hierarchical Task Network (HTN) planners
  - Actually, a specific form of HTN planning known as Ordered Task Decomposition (OTD)
  - OTD planners are useful because of potential for efficiency and expressivity; used in most fielded applications
Objective

- We describe an algorithm for learning such planning knowledge for a certain class of advanced planners known as Hierarchical Task Network (HTN) planners.
  - Actually, a specific form of HTN planning known as Ordered Task Decomposition (OTD)
  - OTD planners are useful because of potential for efficiency and expressivity; used in most fielded applications.
**States** are sets of logical atoms describing the world

*Operators* describe actions, when they may be undertaken, and how they affect the world.

The objective is to find a sequence of actions (a *plan*) that, when executed from a given initial state, will result in a state that satisfies certain given goals.

---

**Example Operator**

```plaintext
:operator
(!load-truck ?pkg ?trk ?loc)

:preconditions
((at ?pkg ?loc)
  (at ?trk ?loc))

:add-effects
((at ?pkg ?loc))

:del-effects
((in ?pkg ?trk))
```
**STRIPS Planning**

- **States** are sets of logical atoms describing the world.
- **Operators** describe actions, when they may be undertaken, and how they affect the world.
- The objective is to find a sequence of actions (a plan) that, when executed from a given initial state, will result in a state that satisfies certain given goals.

**Example Operator**

```plaintext
:operator
  ((!load-truck ?pkg ?trk ?loc))

:preconditions
  ((at ?pkg ?loc)
   (at ?trk ?loc))

:add-effects
  ((at ?pkg ?loc))

:del-effects
  ((in ?pkg ?trk)))
```
**STRIPS Planning**

- **States** are sets of logical atoms describing the world.

- **Operators** describe actions, when they may be undertaken, and how they affect the world.

- The objective is to find a sequence of actions (a plan) that, when executed from a given initial state, will result in a state that satisfies certain given goals.

---

**Example Operator**

```
:operator
  (!load-truck ?pkg ?trk ?loc)

:preconditions
  ((at ?pkg ?loc)
   (at ?trk ?loc))

:add-effects
  ((at ?pkg ?loc))

:del-effects
  ((in ?pkg ?trk))
```
States are sets of logical atoms describing the world.

Operators describe actions, when they may be undertaken, and how they affect the world.

The objective is to find a sequence of actions (a plan) that, when executed from a given initial state, will result in a state that satisfies certain given goals.

---

Example Operator

```lisp
(operator
  ((!load-truck ?pkg ?trk ?loc)
  :preconditions
    ((at ?pkg ?loc)
      (at ?trk ?loc))
  :add-effects
    ((at ?pkg ?loc))
  :del-effects
    ((in ?pkg ?trk)))
```

---

C. Hogg, H. Muñoz-Avila, U. Kuter
**HTN Planning**

- **Tasks** are symbolic representations of activities that should be accomplished.
- **Methods** describe how to decompose complex tasks into sequences of simpler ones.
- The objective is to find a hierarchy of decompositions from a given sequence of tasks down to a sequence of actions that can be executed from the given initial state.

**Example Method**

```prolog
:method
((deliver ?pkg ?loc)
 :preconditions
 ((at ?pkg ?src)
  (at ?trk ?src)
  (same-city ?src ?loc))
 :subtasks
 ((!load-truck ?pkg ?trk ?src)
  (deliver ?pkg ?loc)))
```

C. Hogg, H. Muñoz-Avila, U. Kuter
**Tasks** are symbolic representations of activities that should be accomplished.

**Methods** describe how to decompose complex tasks into sequences of simpler ones.

The objective is to find a hierarchy of decompositions from a given sequence of tasks down to a sequence of actions that can be executed from the given initial state.

**Example Method**

```prolog
:method
((deliver ?pkg ?loc)
 :preconditions
 ((at ?pkg ?src)
  (at ?trk ?src)
  (same-city ?src ?loc))
 :subtasks
 ((!load-truck ?pkg ?trk ?src)
  (deliver ?pkg ?loc)))
```
Tasks are symbolic representations of activities that should be accomplished.

Methods describe how to decompose complex tasks into sequences of simpler ones.

The objective is to find a hierarchy of decompositions from a given sequence of tasks down to a sequence of actions that can be executed from the given initial state.

Example Method

```plaintext
:method
  ((deliver ?pkg ?loc))

:preconditions
  ((at ?pkg ?src)
   (at ?trk ?src)
   (same-city ?src ?loc))

:subtasks
  ((!load-truck ?pkg ?trk ?src)
   (deliver ?pkg ?loc)))
```
- **Tasks** are symbolic representations of activities that should be accomplished.

- **Methods** describe how to decompose complex tasks into sequences of simpler ones.

- The objective is to find a hierarchy of decompositions from a given sequence of tasks down to a sequence of actions that can be executed from the given initial state.

Example Method:

```lisp
:method
  ((deliver ?pkg ?loc)
   :preconditions
     ((at ?pkg ?src)
      (at ?trk ?src)
      (same-city ?src ?loc))
   :subtasks
     (((!load-truck ?pkg ?trk ?src)
        (deliver ?pkg ?loc)))
```
Annotated Tasks

We define the notion of an annotated task as a task with effects (what it means to accomplish the task) and preconditions (when it is possible to accomplish the task).

Example: (deliver ?pkg ?loc) has effect (at ?pkg ?loc)

Given

A STRIPS domain $D$, a collection of annotated tasks $T$, a planning problem $P$ in $D$, and a solution plan $S$ to $P$,

Find

Method definitions for the tasks in $T$ whose subtasks are actions from $S$ or tasks from $T$.
Annotated Tasks

We define the notion of an annotated task as a task with effects (what it means to accomplish the task) and preconditions (when it is possible to accomplish the task)

- Example: (deliver ?pkg ?loc) has effect (at ?pkg ?loc)

Given

A STRIPS domain $D$, a collection of annotated tasks $T$, a planning problem $P$ in $D$, and a solution plan $S$ to $P$,

Find

Method definitions for the tasks in $T$ whose subtasks are actions from $S$ or tasks from $T$
### Annotated Tasks

We define the notion of an *annotated task* as a task with effects (what it means to accomplish the task) and preconditions (when it is possible to accomplish the task).

- Example: `(deliver ?pkg ?loc)` has effect `(at ?pkg ?loc)`

### Given

A STRIPS domain $D$, a collection of annotated tasks $T$, a planning problem $P$ in $D$, and a solution plan $S$ to $P$,

### Find

Method definitions for the tasks in $T$ whose subtasks are actions from $S$ or tasks from $T$
Main Steps of the HTN-MAKER Algorithm

1. Traverse the actions in the input plan, identifying states in which the effects of an annotated task have become satisfied.

2. Regress annotated task effects backward through plan to each previous state where preconditions of annotated task are satisfied.

3. Create a new method for the annotated task. Operators that contribute to the accomplishment of the effects become subtasks of this new method, as may the tasks associated with previously-learned methods.

4. The preconditions of this new method are the union of task preconditions and preconditions of subtasks that are not provided by earlier subtasks.

5. Learned methods are stored so that their tasks may be used as subtasks in methods that will be learned later.
Main Steps of the HTN-MAKER Algorithm

1. Traverse the actions in the input plan, identifying states in which the effects of an annotated task have become satisfied

2. Regress annotated task effects backward through plan to each previous state where preconditions of annotated task are satisfied

3. Create a new method for the annotated task. Operators that contribute to the accomplishment of the effects become subtasks of this new method, as may the tasks associated with previously-learned methods

4. The preconditions of this new method are the union of task preconditions and preconditions of subtasks that are not provided by earlier subtasks

5. Learned methods are stored so that their tasks may be used as subtasks in methods that will be learned later
Main Steps of the HTN-MAKER Algorithm

1. Traverse the actions in the input plan, identifying states in which the effects of an annotated task have become satisfied.

2. Regress annotated task effects backward through plan to each previous state where preconditions of annotated task are satisfied.

3. Create a new method for the annotated task. Operators that contribute to the accomplishment of the effects become subtasks of this new method, as may the tasks associated with previously-learned methods.

4. The preconditions of this new method are the union of task preconditions and preconditions of subtasks that are not provided by earlier subtasks.

5. Learned methods are stored so that their tasks may be used as subtasks in methods that will be learned later.
Main Steps of the HTN-MAKER Algorithm

1. Traverse the actions in the input plan, identifying states in which the effects of an annotated task have become satisfied

2. Regress annotated task effects backward through plan to each previous state where preconditions of annotated task are satisfied

3. Create a new method for the annotated task. Operators that contribute to the accomplishment of the effects become subtasks of this new method, as may the tasks associated with previously-learned methods

4. The preconditions of this new method are the union of task preconditions and preconditions of subtasks that are not provided by earlier subtasks

5. Learned methods are stored so that their tasks may be used as subtasks in methods that will be learned later
Main Steps of the HTN-MAKER Algorithm

1. Traverse the actions in the input plan, identifying states in which the effects of an annotated task have become satisfied

2. Regress annotated task effects backward through plan to each previous state where preconditions of annotated task are satisfied

3. Create a new method for the annotated task. Operators that contribute to the accomplishment of the effects become subtasks of this new method, as may the tasks associated with previously-learned methods

4. The preconditions of this new method are the union of task preconditions and preconditions of subtasks that are not provided by earlier subtasks

5. Learned methods are stored so that their tasks may be used as subtasks in methods that will be learned later
Example

Input is initial state (S0), plan, and annotated tasks
Use initial state and plan to fill in intermediate states

<table>
<thead>
<tr>
<th>STATE S0:</th>
<th>(load P T L1)</th>
<th>(drive T L1 L2 C)</th>
<th>(unload P T L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre:</td>
<td>(at P L1)</td>
<td>(at T L1)</td>
<td>(at T L2)</td>
</tr>
<tr>
<td>(city C)</td>
<td>(at P L1)</td>
<td>(in-city L1 C)</td>
<td>(in P T)</td>
</tr>
<tr>
<td>(location L1)</td>
<td>(at T L1)</td>
<td>(in-city L2 C)</td>
<td>(at T L2)</td>
</tr>
<tr>
<td>(location L2)</td>
<td>(at T L1)</td>
<td>eff+:</td>
<td>(in P T)</td>
</tr>
<tr>
<td>(truck T)</td>
<td>(at P L1)</td>
<td>eff+:</td>
<td>(at P L2)</td>
</tr>
<tr>
<td>(package P)</td>
<td>(at P T)</td>
<td>eff-:</td>
<td>(at T L2)</td>
</tr>
<tr>
<td>(in-city L1 C)</td>
<td>(at P L1)</td>
<td>eff-:</td>
<td>(in P T)</td>
</tr>
<tr>
<td>(in-city L2 C)</td>
<td>(at P L1)</td>
<td>(at T L1)</td>
<td>(at T L2)</td>
</tr>
<tr>
<td>(at T L1)</td>
<td>(at T L1)</td>
<td>(at T L2)</td>
<td>(at P L2)</td>
</tr>
<tr>
<td>(at P L1)</td>
<td>(at P L1)</td>
<td>(at T L1)</td>
<td>(at P L2)</td>
</tr>
</tbody>
</table>

Task: (deliver ?p ?l)
pre:
eff+: (at ?p ?l)
eff-:
The effects of \((\text{deliver} \ ?p \ ?l)\) become true in S3 with \(?p = P, \ ?l = L2\), so learn some methods starting with the previous.

Initial method has effects of annotated task as preconditions, no subtasks.
The effects of (deliver ?p ?l) become true in S3 with ?p = P, ?l = L2, so learn some methods starting with the previous.

Initial method has effects of annotated task as preconditions, no subtasks.
Example (4)

The (!unload P T L2) operator fulfills the outstanding precondition, so remove it, add the preconditions of the operator to the method, and make it a subtask. This is now a method describing how to deliver a package from a certain set of states.
The `(!unload P T L2)` operator fulfills the outstanding precondition, so remove it, add the preconditions of the operator to the method, and make it a subtask.

This is now a method describing how to deliver a package from a certain set of states.
Now we go back one step further in the plan, starting again with an empty method.
The method that we already learned fulfills the outstanding precondition (through the minimal effects of its annotated task), so remove it, add preconditions of method, and make it a subtask.
The (!drive T L1 L2 C) operator fulfills an outstanding precondition, so remove it, add the preconditions of the operator to the method, and make it a subtask.

This is now a method describing how to deliver a package from an additional set of states.
The (!drive T L1 L2 C) operator fulfills an outstanding precondition, so remove it, add the preconditions of the operator to the method, and make it a subtask.

This is now a method describing how to deliver a package from an additional set of states.
Now we go back to the first step in the plan.
The new previously learned method is useful in the same way.
The (!load P T L1) operator fulfills a precondition, so we add it as well.

This is a third method, allowing us to accomplish the task from yet another set of states.
The (!load P T L1) operator fulfills a precondition, so we add it as well.

This is a third method, allowing us to accomplish the task from yet another set of states.
Theoretical Results

- We say that a task $t$ can be made equivalent to the goals $G$ of a classical planning problem by making $G$ the annotated effects of $t$
- Planning with methods learned by HTN-MAKER is sound, relative to the goals equivalent to the tasks
- There exists a finite number of input problems in a domain such that planning with methods learned by HTN-MAKER on them is complete, relative to the annotated tasks provided
- HTN Planning with a learned domain is more expressive than STRIPS planning
  - Classically-partitionable problems, which are sequences of (state, goal) pairs, are an example of a class of problems that can be solved using learned methods but cannot be expressed in classical planning
We say that a task $t$ can be made equivalent to the goals $G$ of a classical planning problem by making $G$ the annotated effects of $t$.

Planning with methods learned by HTN-MAKER is sound, relative to the goals equivalent to the tasks.

There exists a finite number of input problems in a domain such that planning with methods learned by HTN-MAKER on them is complete, relative to the annotated tasks provided.

HTN Planning with a learned domain is more expressive than STRIPS planning.

Classically-partitionable problems, which are sequences of (state, goal) pairs, are an example of a class of problems that can be solved using learned methods but cannot be expressed in classical planning.
Theoretical Results

- We say that a task $t$ can be made equivalent to the goals $G$ of a classical planning problem by making $G$ the annotated effects of $t$.
- Planning with methods learned by HTN-MAKER is *sound*, relative to the goals equivalent to the tasks.
- There exists a finite number of input problems in a domain such that planning with methods learned by HTN-MAKER on them is *complete*, relative to the annotated tasks provided.
- HTN Planning with a learned domain is more *expressive* than STRIPS planning.
  - Classically-partitionable problems, which are sequences of (state, goal) pairs, are an example of a class of problems that can be solved using learned methods but cannot be expressed in classical planning.
Theoretical Results

- We say that a task $t$ can be made equivalent to the goals $G$ of a classical planning problem by making $G$ the annotated effects of $t$
- Planning with methods learned by HTN-MAKER is *sound*, relative to the goals equivalent to the tasks
- There exists a finite number of input problems in a domain such that planning with methods learned by HTN-MAKER on them is *complete*, relative to the annotated tasks provided
- HTN Planning with a learned domain is more *expressive* than STRIPS planning

  *Classically-partitionable* problems, which are sequences of (state, goal) pairs, are an example of a class of problems that can be solved using learned methods but cannot be expressed in classical planning
Theoretical Results

- We say that a task $t$ can be made equivalent to the goals $G$ of a classical planning problem by making $G$ the annotated effects of $t$
- Planning with methods learned by HTN-MAKER is *sound*, relative to the goals equivalent to the tasks
- There exists a finite number of input problems in a domain such that planning with methods learned by HTN-MAKER on them is *complete*, relative to the annotated tasks provided
- HTN Planning with a learned domain is more *expressive* than STRIPS planning
  - *Classically-partitionable* problems, which are sequences of (state, goal) pairs, are an example of a class of problems that can be solved using learned methods but cannot be expressed in classical planning
Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains

- These are common benchmarks from planning competitions and the planning literature
- Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods
- Increase size of training set linearly from 1 to 75 problems
- Results averaged over 5 random problem distributions
- Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not
Evaluation

- Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains
  - These are common benchmarks from planning competitions and the planning literature

- Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods

- Increase size of training set linearly from 1 to 75 problems

- Results averaged over 5 random problem distributions

- Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not
Evaluation

- Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains
  - These are common benchmarks from planning competitions and the planning literature
- Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods
- Increase size of training set linearly from 1 to 75 problems
- Results averaged over 5 random problem distributions
- Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not
Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains

These are common benchmarks from planning competitions and the planning literature.

Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods.

Increase size of training set linearly from 1 to 75 problems.

Results averaged over 5 random problem distributions.

Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not.
Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains

- These are common benchmarks from planning competitions and the planning literature

Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods

Increase size of training set linearly from 1 to 75 problems

Results averaged over 5 random problem distributions

Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not
Experiments measuring coverage in Logistics-Transportation, Blocks-World, and Satellite domains

- These are common benchmarks from planning competitions and the planning literature

Begin with HTN domain containing no methods, learn from solutions to a set of training problems, attempt to solve test problems with the learned methods

Increase size of training set linearly from 1 to 75 problems

Results averaged over 5 random problem distributions

Logistics-Transportation and Satellite show rapid convergence to nearly complete domains, Blocks-World does not
Results

C. Hogg, H. Muñoz-Avila, U. Kuter

HTN-MAKER
Since paper submission we have been working on planning efficiency results

- Currently only have results for Logistics-Transportation
- For small problems, FastForward is much faster than HTN planner with learned domain
- For large problems, advantages of HTN planning overcome lack of focus in domain
Since paper submission we have been working on planning efficiency results

Currently only have results for Logistics-Transportation

For small problems, FastForward is much faster than HTN planner with learned domain

For large problems, advantages of HTN planning overcome lack of focus in domain
Since paper submission we have been working on planning efficiency results.

Currently only have results for Logistics-Transportation.

For small problems, FastForward is much faster than HTN planner with learned domain.

For large problems, advantages of HTN planning overcome lack of focus in domain.
Since paper submission we have been working on planning efficiency results.

Currently only have results for Logistics-Transportation.

For small problems, FastForward is much faster than HTN planner with learned domain.

For large problems, advantages of HTN planning overcome lack of focus in domain.
Results (2)
We have described a new algorithm, HTN-MAKER, for learning HTN domain knowledge from classical planning problems and their solutions.

We have presented theoretical results showing that:
- The methods learned by HTN-MAKER are sound and complete relative to the set of goals for which annotated tasks are provided.
- The learned methods can be used to solve problems that could not be expressed using the classical planning knowledge from which they were learned.

Our experiments in three well-known planning domains demonstrated that HTN-MAKER converged to a set of HTN methods that could solve nearly all problems in the domain.
Conclusions

- We have described a new algorithm, HTN-MAKER, for learning HTN domain knowledge from classical planning problems and their solutions.

- We have presented theoretical results showing that:
  - The methods learned by HTN-MAKER are sound and complete relative to the set of goals for which annotated tasks are provided.
  - The learned methods can be used to solve problems that could not be expressed using the classical planning knowledge from which they were learned.

- Our experiments in three well-known planning domains demonstrated that HTN-MAKER converged to a set of HTN methods that could solve nearly all problems in the domain.
Conclusions

- We have described a new algorithm, HTN-MAKER, for learning HTN domain knowledge from classical planning problems and their solutions.
- We have presented theoretical results showing that:
  - The methods learned by HTN-MAKER are sound and complete relative to the set of goals for which annotated tasks are provided.
  - The learned methods can be used to solve problems that could not be expressed using the classical planning knowledge from which they were learned.
- Our experiments in three well-known planning domains demonstrated that HTN-MAKER converged to a set of HTN methods that could solve nearly all problems in the domain.
We have described a new algorithm, HTN-MAKER, for learning HTN domain knowledge from classical planning problems and their solutions.

We have presented theoretical results showing that:

- The methods learned by HTN-MAKER are sound and complete relative to the set of goals for which annotated tasks are provided.
- The learned methods can be used to solve problems that could not be expressed using the classical planning knowledge from which they were learned.

Our experiments in three well-known planning domains demonstrated that HTN-MAKER converged to a set of HTN methods that could solve nearly all problems in the domain.
We have described a new algorithm, HTN-MAKER, for learning HTN domain knowledge from classical planning problems and their solutions.

We have presented theoretical results showing that:
- The methods learned by HTN-MAKER are sound and complete relative to the set of goals for which annotated tasks are provided.
- The learned methods can be used to solve problems that could not be expressed using the classical planning knowledge from which they were learned.

Our experiments in three well-known planning domains demonstrated that HTN-MAKER converged to a set of HTN methods that could solve nearly all problems in the domain.
Conclusions (2)

Future Work
- Refine methods through merging or Reinforcement Learning-based selection
- Study effect of providing more or less information in annotated tasks

Thank You
- Questions?