Learning Hierarchical Problem Networks for Knowledge-Based Planning

Pat Langley

Institute for the Study of
Learning and Expertise
Palo Alto, California

Thanks to Howie Shrobe, Boris Katz, Gary Borchardt, Sue Felshin, Mohan Sridharan, and Ed Katz for useful ideas and discussions. This research was supported by Grant No. N00014-20-1-2643 from ONR, which is not responsible for its contents.

Reasoning, Search, and Knowledge

Classic accounts of intelligence in humans and machines rely on three complementary elements:

- *Reasoning – Drawing conclusions by composing elements*
- *Search – Finding solutions in large problem spaces*
- *Knowledge – Elements for reasoning, guidance for search*

Early AI saw rapid progress on reasoning / search, but major applications awaited expert systems.

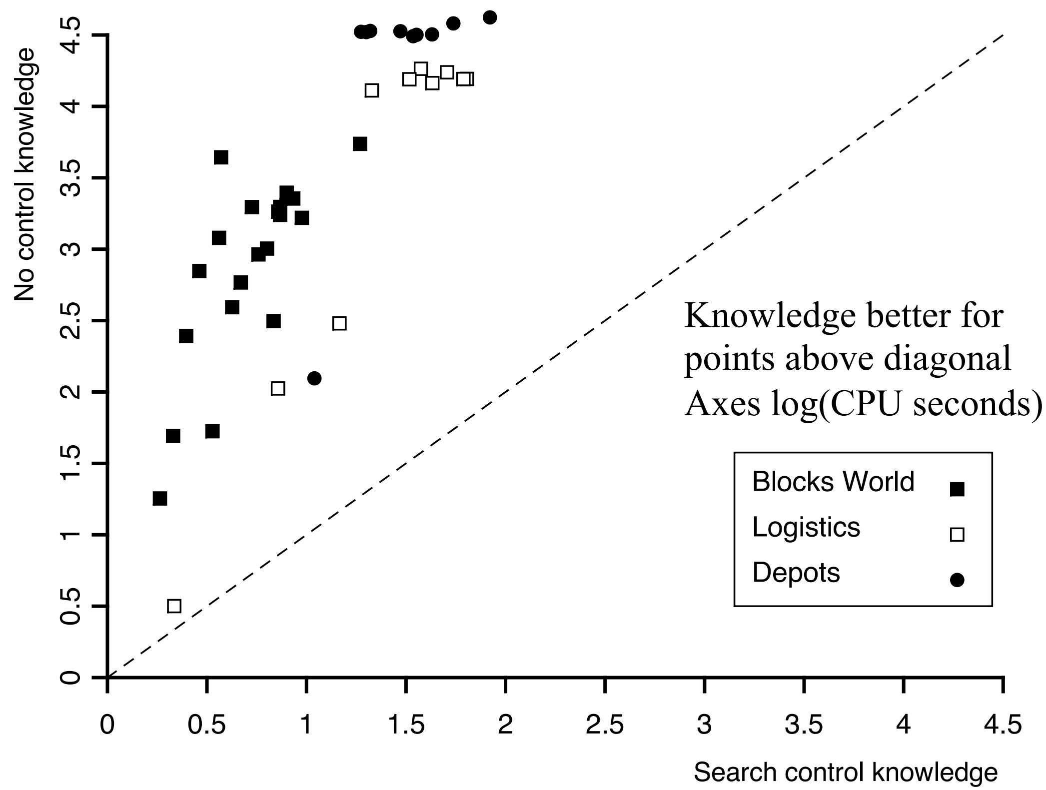
Yet knowledge was difficult to extract, which led in turn to the advent of *machine learning*.

The Need for Planning Expertise

Machine learning has seen decades of progress on classification and reactive control, but far less on *planning*.

This technology has improved, but it still benefits greatly from handcrafted expertise.

In this talk, I present a new approach to learning knowledge for planning from sample solutions.



Encoding Planning Knowledge

A recurring theme in AI and psychology is the ***decomposition*** of complex problems into simpler ones.

This idea is central to logic programming, but it also appears in planning as:

- *Hierarchical task networks (Nau et al., 2003)*
- *Teleoreactive logic programs (Choi & Langley, 2005)*
- *Hierarchical goal networks (Shivashankar et al., 2012)*

Here I focus on acquiring knowledge in a new framework – ***hierarchical problem networks*** – that offers advantages.

Hierarchical Problem Networks

A hierarchical problem network encodes planning knowledge as a set of *methods*, each of which includes:

- A generic ***goal*** that the method achieves
- ***State conditions*** under which it can apply
- ***Goal conditions*** under which it should ***not*** apply
- An ***operator*** that will achieve the goal
- A ***subproblem*** based on the operator's conditions

An HPN specifies how to decompose an entire problem – a set of goals – into ordered subproblems.

HPN Methods for Logistics

```
((at ?o1 ?l3))
 conditions:   ((object ?o1) (truck ?t1) (location ?l3) (location ?l1)
                (in-city ?l3 ?c1) (in-city ?l1 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (unload-truck ?o1 ?t1 ?l3)
 subproblem:   ((at ?t1 ?l3) (in ?o1 ?t1))
((at ?t1 ?l1))
 conditions:   ((truck ?t1) (location ?l3) (location ?l1)
                (city ?c1) (in-city ?l3 ?c1) (in-city ?l1 ?c1)
                (in-city ?l2 ?c1) (at ?t1 ?l3))
 operator:     (drive-truck ?t1 ?l3 ?l1 ?c1)
 subproblem:   ((at ?t1 ?l3))
 unless-goals: ((in ?o ?t1)))
((in ?o1 ?t1))
 conditions:   ((object ?o1) (truck ?t1) (location ?l1) (location ?l3)
                (in-city ?l1 ?c1) (in-city ?l3 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (load-truck ?o1 ?t1 ?l1)
 subproblem:   ((at ?t1 ?l1) (at ?o1 ?l1))
((in ?o1 ?t1)
 :conditions   ((object ?o1) (truck ?t1) (location ?l1) (airport ?l1)
                (location ?l2) (location ?l3) (in-city ?l1 ?c1) (in-city ?l2 ?c1)
                (in-city ?l3 ?c2) (at ?t1 ?l2) (at ?o1 ?l3))
 :operator     (load-truck ?o1 ?t1 ?l1)
 :subproblem   ((at ?t1 ?l1) (at ?o1 ?l1))
```

HPN Operators for Logistics

```
((unload-truck ?o ?t ?l)
 :conditions ((object ?o)(truck ?t)(location ?l)(at ?t ?l)(in ?o ?t))
 :effects    ((at ?o ?l)(not (in ?o ?t))))
((drive-truck ?t ?l1 ?l2 ?c)
 :conditions ((truck ?t)(location ?l1)(location ?l2)(city ?c)
              (in-city ?l1 ?c)(in-city ?l2 ?c)(at ?t ?l1))
 :effects    ((at ?t ?l2)(not (at ?t ?l1))))
((load-truck ?o ?t ?l)
 :conditions ((object ?o)(truck ?t)(location ?l)(at ?t ?l)(at ?o ?l))
 :effects    ((in ?o ?t)(not (at ?o ?l))))
```

Hierarchical Problem Decomposition

Given goals to achieve and an initial state, a planner can use HPN methods to:

- *Iteratively examine the topmost problem on a stack and place new subproblems above it.*
- *Recursively decompose a problem into subproblems to give an operator sequence that achieves the problem's goals.*
- *Pursue AND/OR search through a space of decompositions defined by methods and problem goals.*

Problem solving is similar to that for HTNs and HGNs, but it decomposes ***problems*** rather than tasks or goals.

The Task of Learning HPNs

We can specify the task of learning an HPN in terms of inputs and outputs:

- ***Given:*** *Domain operators with conditional effects of actions;*
- ***Given:*** *Training tasks with initial states and conjunctive goals;*
- ***Given:*** *A hierarchical plan for each task that achieves its goals;*
- ***Find:*** *An HPN that solves training tasks efficiently and that generalizes well to new cases.*

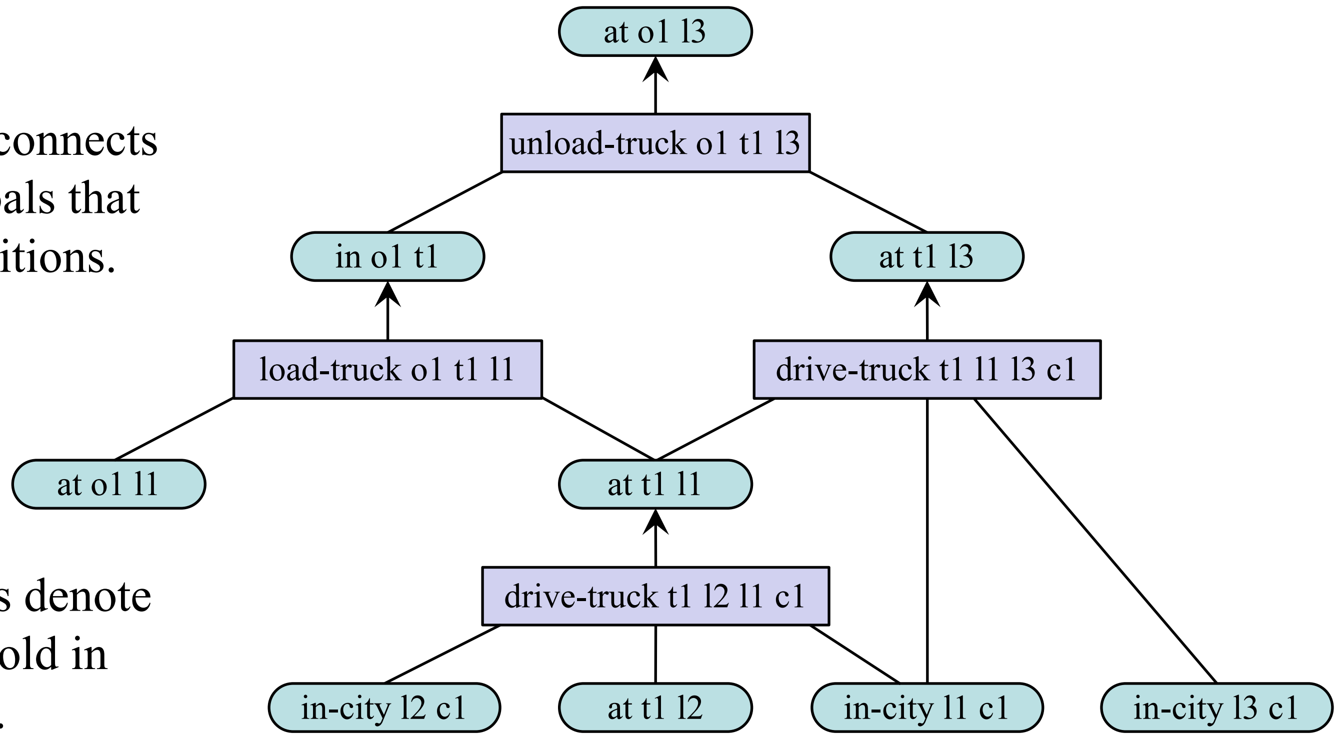
The learned HPN should find plans with little or no search and improvement should occur rapidly.

Inputs to HPN Learning

Consider a simple hierarchical plan for a one-goal logistics problem.

Each operator connects a goal to subgoals that satisfy its conditions.

Terminal nodes denote relations that hold in the initial state.

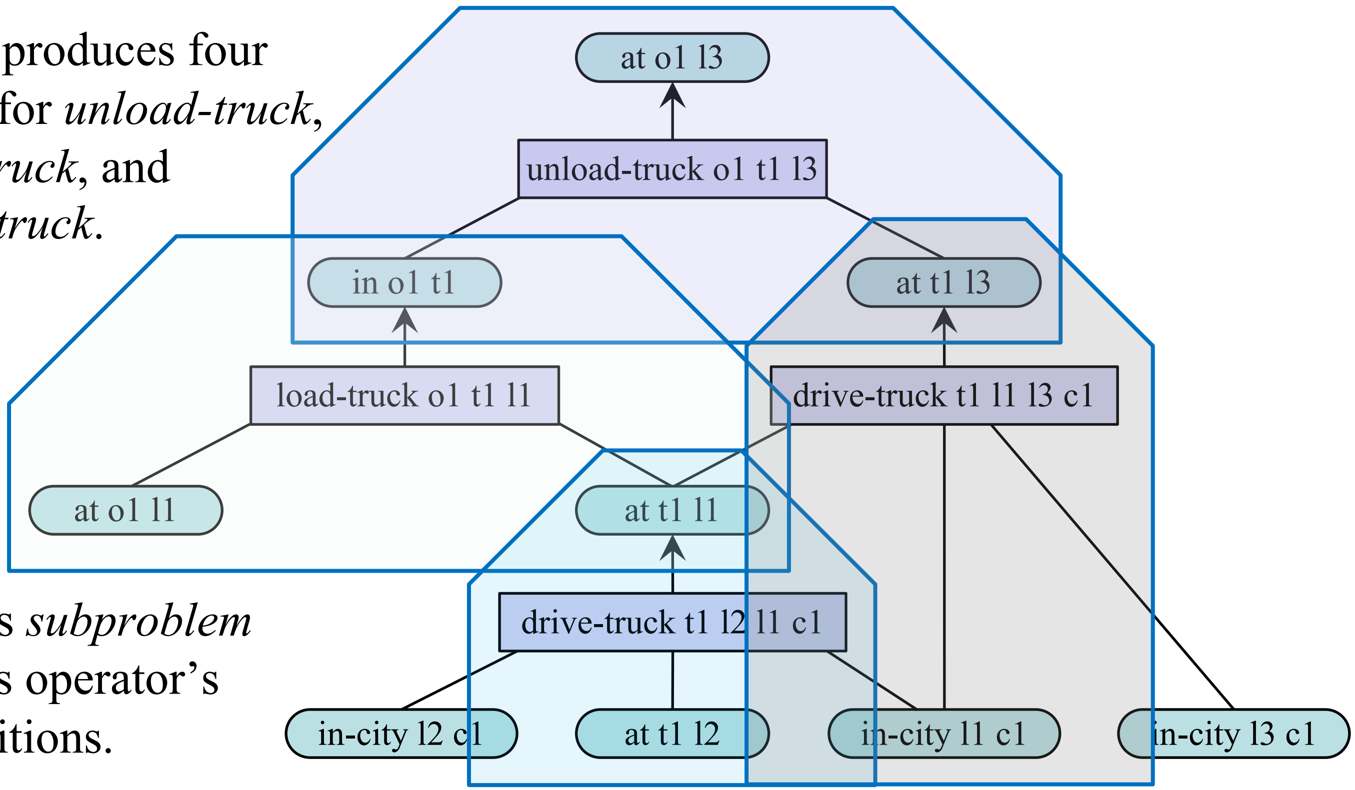


This sample plan provides one training problem for HPN learning.

Identifying HPN Structure

For HPN structure, we create one method per sample decomposition.

Here learning produces four methods, one for *unload-truck*, one for *load-truck*, and two for *drive-truck*.



Each method's *subproblem* comes from its operator's *dynamic* conditions.

Goals serve as method heads, so nonterminal symbols are unneeded.

HPN Methods for Logistics

```
((at ?o1 ?l3))
 conditions:   ((object ?o1) (truck ?t1) (location ?l3) (location ?l1)
                (in-city ?l3 ?c1) (in-city ?l1 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (unload-truck ?o1 ?t1 ?l3)
 subproblem:   ((at ?t1 ?l3) (in ?o1 ?t1))
((at ?t1 ?l1))
 conditions:   ((truck ?t1) (location ?l3) (location ?l1)
                (city ?c1) (in-city ?l3 ?c1) (in-city ?l1 ?c1)
                (in-city ?l2 ?c1) (at ?t1 ?l3))
 operator:     (drive-truck ?t1 ?l3 ?l1 ?c1)
 subproblem:   ((at ?t1 ?l3))
 unless-goals: ((in ?o ?t1)))
((in ?o1 ?t1))
 conditions:   ((object ?o1) (truck ?t1) (location ?l1) (location ?l3)
                (in-city ?l1 ?c1) (in-city ?l3 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (load-truck ?o1 ?t1 ?l1)
 subproblem:   ((at ?t1 ?l1) (at ?o1 ?l1))
((in ?o1 ?t1)
 :conditions   ((object ?o1) (truck ?t1) (location ?l1) (airport ?l1)
                (location ?l2) (location ?l3) (in-city ?l1 ?c1) (in-city ?l2 ?c1)
                (in-city ?l3 ?c2) (at ?t1 ?l2) (at ?o1 ?l3))
 :operator     (load-truck ?o1 ?t1 ?l1)
 :subproblem   ((at ?t1 ?l1) (at ?o1 ?l1))
```

Each method's *subproblem* comes from its operator's *dynamic* conditions.

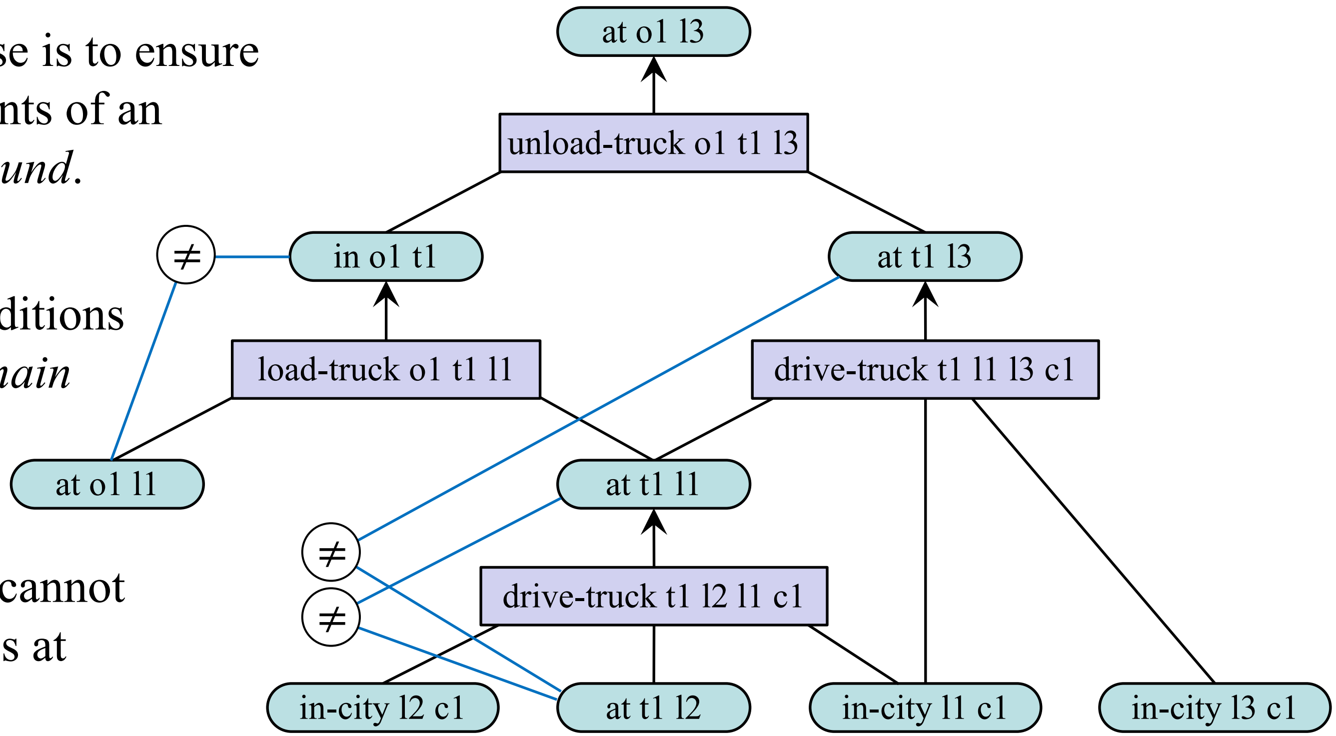
Inferring State Conditions

Finding state conditions relies on much simpler analysis than ILP or EBL.

The key purpose is to ensure that all arguments of an operator are *bound*.

Basic state conditions come from *domain constraints*.

E.g., an object cannot be in two places at the same time.



Here *(at T1 L2)* becomes a condition because it conflicts with *(at T1 L3)*.

HPN Methods for Logistics

```
((at ?o1 ?l3))
 conditions:   ((object ?o1) (truck ?t1) (location ?l3) (location ?l1)
                (in-city ?l3 ?c1) (in-city ?l1 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (unload-truck ?o1 ?t1 ?l3)
 subproblem:   ((at ?t1 ?l3) (in ?o1 ?t1))
((at ?t1 ?l1))
 conditions:   ((truck ?t1) (location ?l3) (location ?l1)
                (city ?c1) (in-city ?l3 ?c1) (in-city ?l1 ?c1)
                (in-city ?l2 ?c1) (at ?t1 ?l3))
 operator:     (drive-truck ?t1 ?l3 ?l1 ?c1)
 subproblem:   ((at ?t1 ?l3))
 unless-goals: ((in ?o ?t1)))
((in ?o1 ?t1))
 conditions:   ((object ?o1) (truck ?t1) (location ?l1) (location ?l3)
                (in-city ?l1 ?c1) (in-city ?l3 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (load-truck ?o1 ?t1 ?l1)
 subproblem:   ((at ?t1 ?l1) (at ?o1 ?l1))
((in ?o1 ?t1)
 :conditions   ((object ?o1) (truck ?t1) (location ?l1) (airport ?l1)
                (location ?l2) (location ?l3) (in-city ?l1 ?c1) (in-city ?l2 ?c1)
                (in-city ?l3 ?c2) (at ?t1 ?l2) (at ?o1 ?l3))
 :operator     (load-truck ?o1 ?t1 ?l1)
 :subproblem   ((at ?t1 ?l1) (at ?o1 ?l1))
```

Basic state conditions come from *domain constraints*.

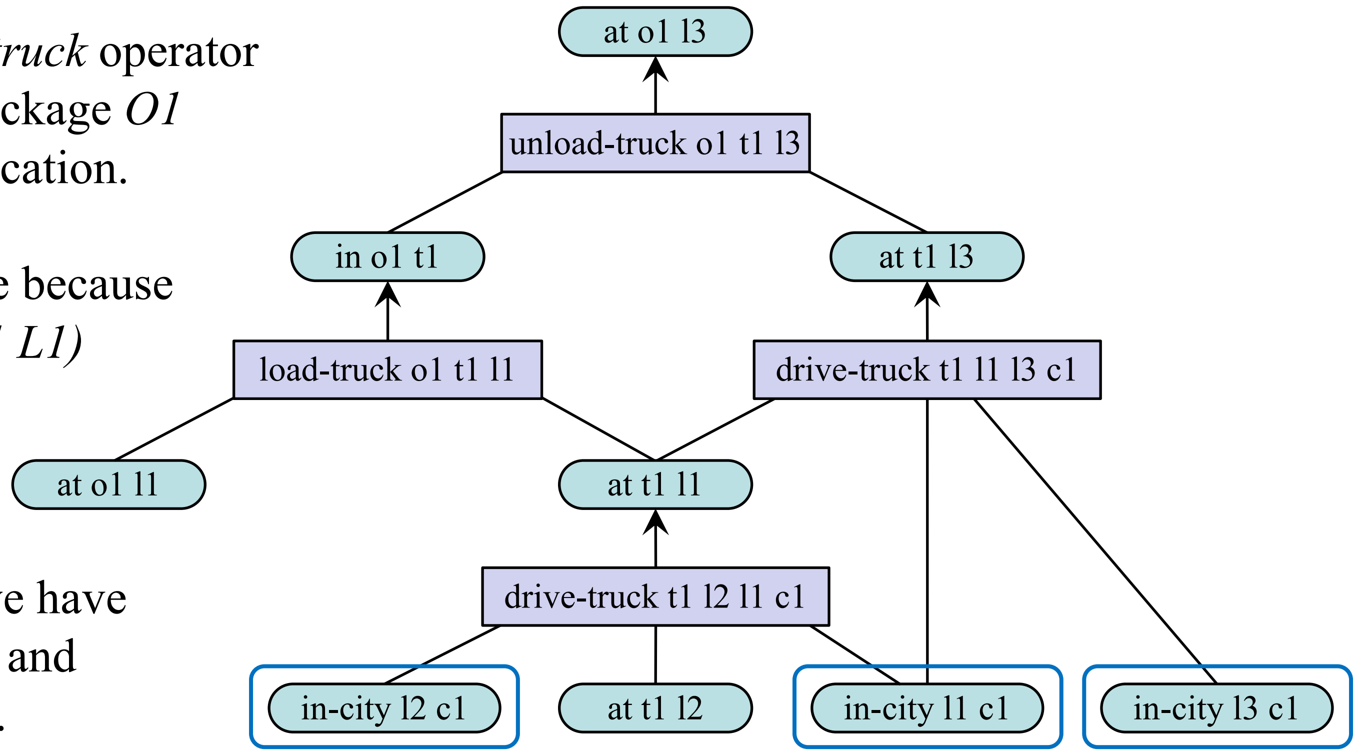
Extending State Conditions

We may need to include *static* literals to ensure proper argument bindings.

Here the *load-truck* operator picks up the package *O1* at its current location.

This is possible because we have *(at O1 L1)* and *(at T1 L2)*.

And because we have *(in-city L1 C1)* and *(in-city L2 C1)*.



We can find these two static conditions by chaining out from L1 and L2.

HPN Methods for Logistics

```
((at ?o1 ?l3))
 conditions:   ((object ?o1) (truck ?t1) (location ?l3) (location ?l1)
                (in-city ?l3 ?c1) (in-city ?l1 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (unload-truck ?o1 ?t1 ?l3)
 subproblem:   ((at ?t1 ?l3) (in ?o1 ?t1))
((at ?t1 ?l1))
 conditions:   ((truck ?t1) (location ?l3) (location ?l1)
                (city ?c1) (in-city ?l3 ?c1) (in-city ?l1 ?c1)
                (in-city ?l2 ?c1) (at ?t1 ?l3))
 operator:     (drive-truck ?t1 ?l3 ?l1 ?c1)
 subproblem:   ((at ?t1 ?l3))
 unless-goals: ((in ?o ?t1)))
((in ?o1 ?t1))
 conditions:   ((object ?o1) (truck ?t1) (location ?l1) (location ?l3)
                (in-city ?l1 ?c1) (in-city ?l3 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (load-truck ?o1 ?t1 ?l1)
 subproblem:   ((at ?t1 ?l1) (at ?o1 ?l1))
((in ?o1 ?t1)
 :conditions   ((object ?o1) (truck ?t1) (location ?l1) (airport ?l1)
                (location ?l2) (location ?l3) (in-city ?l1 ?c1) (in-city ?l2 ?c1)
                (in-city ?l3 ?c2) (at ?t1 ?l2) (at ?o1 ?l3))
 :operator     (load-truck ?o1 ?t1 ?l1)
 :subproblem   ((at ?t1 ?l1) (at ?o1 ?l1))
```

Chaining adds static conditions to ensure proper bindings.

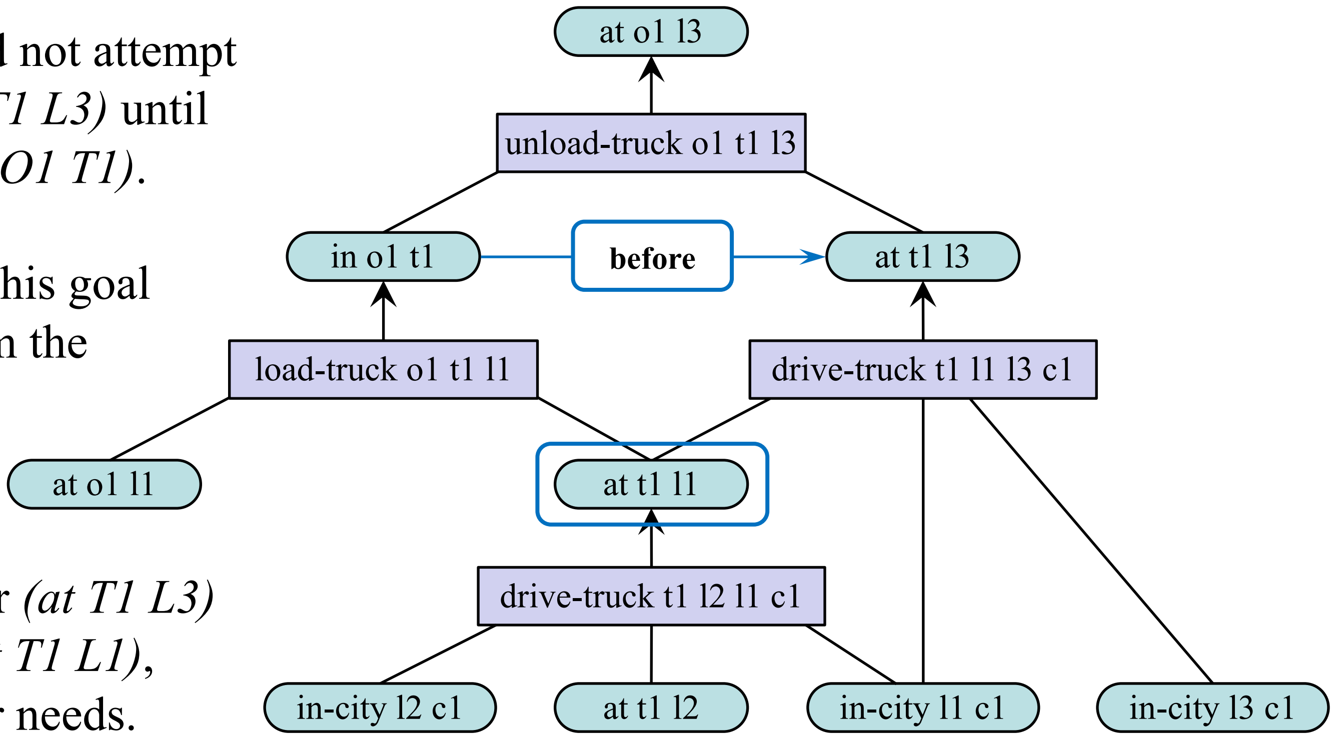
Finding Goal Conditions

We must also find goal conditions to constrain the order of method selection.

Here we should not attempt to achieve *(at T1 L3)* until we achieve *(in O1 T1)*.

We can detect this goal interaction from the two subplans.

The subplan for *(at T1 L3)* will clobber *(at T1 L1)*, which the other needs.



Thus, we must add goal condition *(in ?O1 ?T1)* to the *(at ?T1 ?L3)* method.

HPN Methods for Logistics

```
((at ?o1 ?l3))
 conditions:   ((object ?o1) (truck ?t1) (location ?l3) (location ?l1)
                (in-city ?l3 ?c1) (in-city ?l1 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (unload-truck ?o1 ?t1 ?l3)
 subproblem:   ((at ?t1 ?l3) (in ?o1 ?t1))
((at ?t1 ?l1))
 conditions:   ((truck ?t1) (location ?l3) (location ?l1)
                (city ?c1) (in-city ?l3 ?c1) (in-city ?l1 ?c1)
                (in-city ?l2 ?c1) (at ?t1 ?l3))
 operator:     (drive-truck ?t1 ?l3 ?l1 ?c1)
 subproblem:   ((at ?t1 ?l3))
 unless-goals: ((in ?o ?t1)))
((in ?o1 ?t1))
 conditions:   ((object ?o1) (truck ?t1) (location ?l1) (location ?l3)
                (in-city ?l1 ?c1) (in-city ?l3 ?c1) (at ?t1 ?l3) (at ?o1 ?l1))
 operator:     (load-truck ?o1 ?t1 ?l1)
 subproblem:   ((at ?t1 ?l1) (at ?o1 ?l1))
((in ?o1 ?t1)
 :conditions   ((object ?o1) (truck ?t1) (location ?l1) (airport ?l1)
                (location ?l2) (location ?l3) (in-city ?l1 ?c1) (in-city ?l2 ?c1)
                (in-city ?l3 ?c2) (at ?t1 ?l2) (at ?o1 ?l3))
 :operator     (load-truck ?o1 ?t1 ?l1)
 :subproblem   ((at ?t1 ?l1) (at ?o1 ?l1))
```

Empirical Evaluation

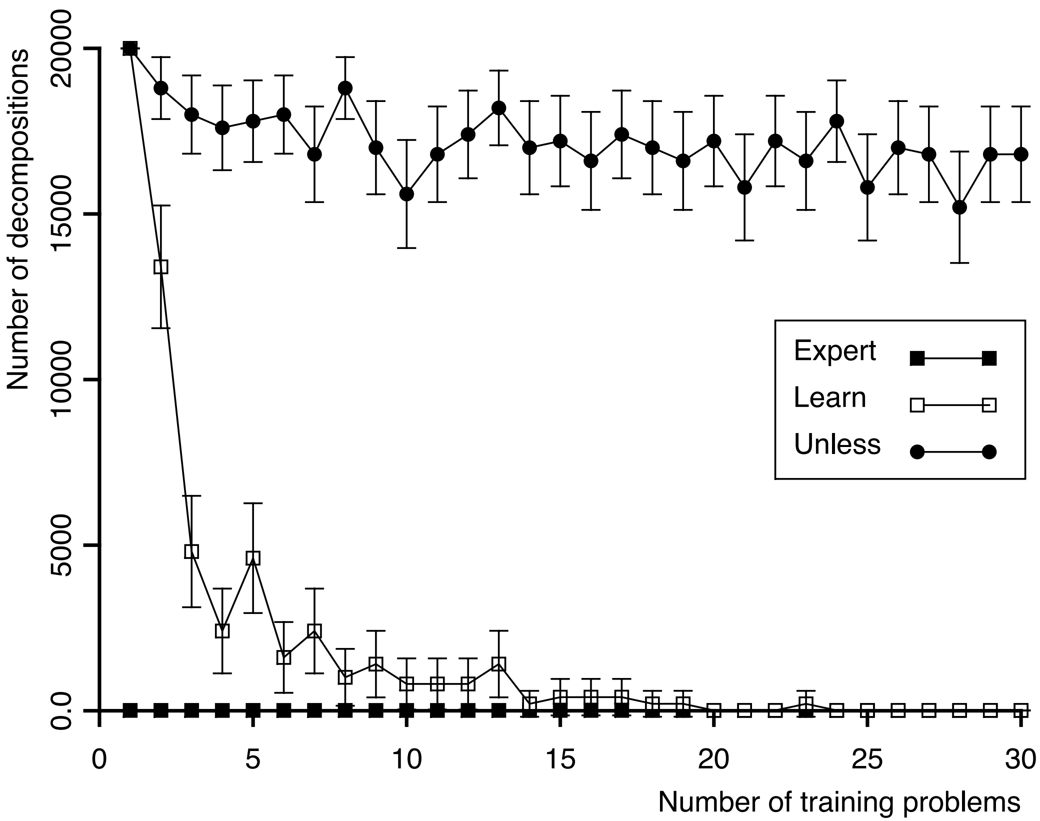
To show effectiveness, we implemented a learning system – HPNL – and ran experiments with:

- *Three planning domains (Blocks World, Logistics, Depots)*
- *30 distinct problems for each domain / solutions 4 to 21 steps*
- *A limit of 30 to 50 plan steps and 20,000 decompositions*
- *Each problem used first for testing and then for training*
- *Decompositions / CPU seconds as dependent measures*
- *Averaged results over 100 random problem orders*

We focused on learning rate, learned vs. handcrafted expertise, and benefits of goal conditions.

HPNL Results on Blocks World

Number of decompositions needed to solve problems in the Blocks World.



Mastery occurs very rapidly in this domain.

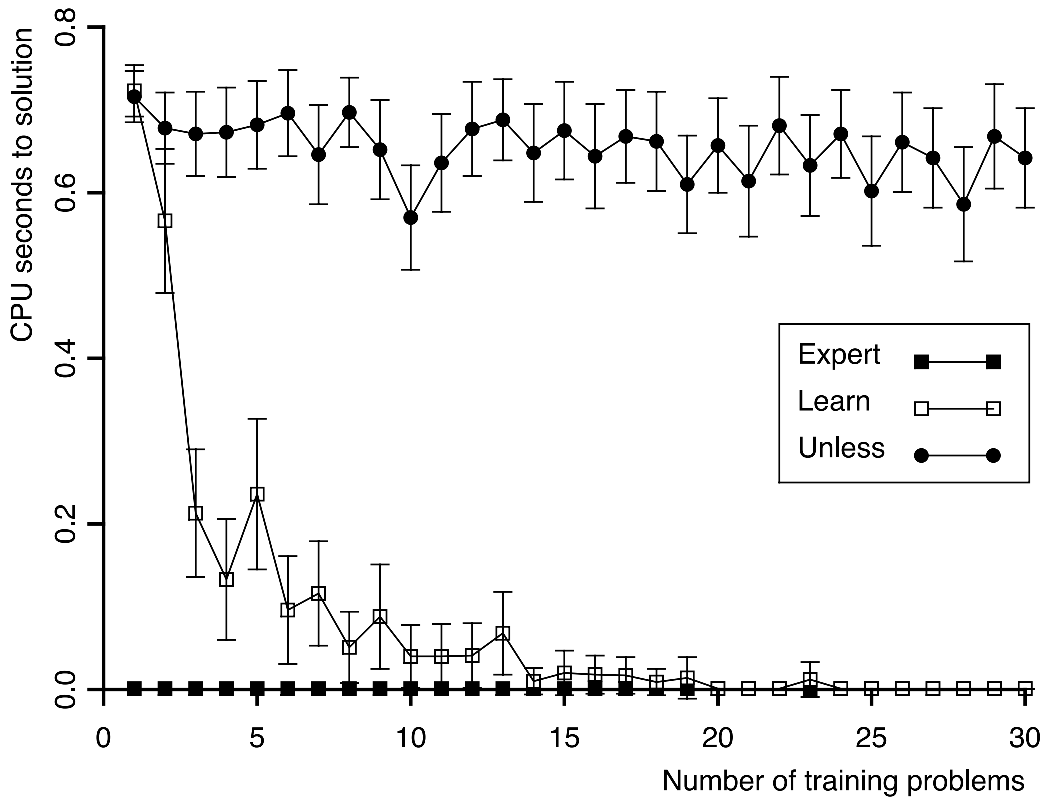
*But **only** if we include goal conditions.*

Each curve shows means with 95% confidence intervals over 100 orders.

More Results on Blocks World

Processing (CPU seconds) needed to solve problems in the Blocks World.

*HPNL does **not** seem to have a utility problem.*

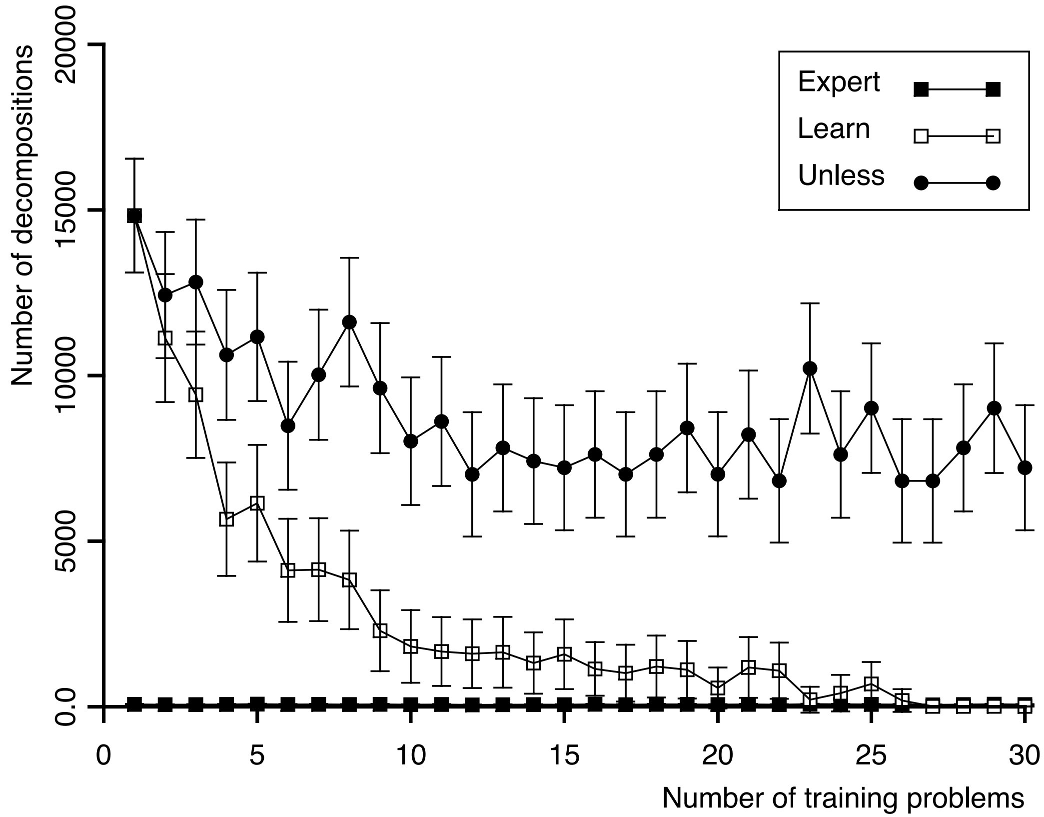


Each curve shows means with 95% confidence intervals over 100 orders.

HPNL Results on Logistics

Number of decompositions needed to solve problems in Logistics planning.

*This domain is **harder** than Blocks World.*



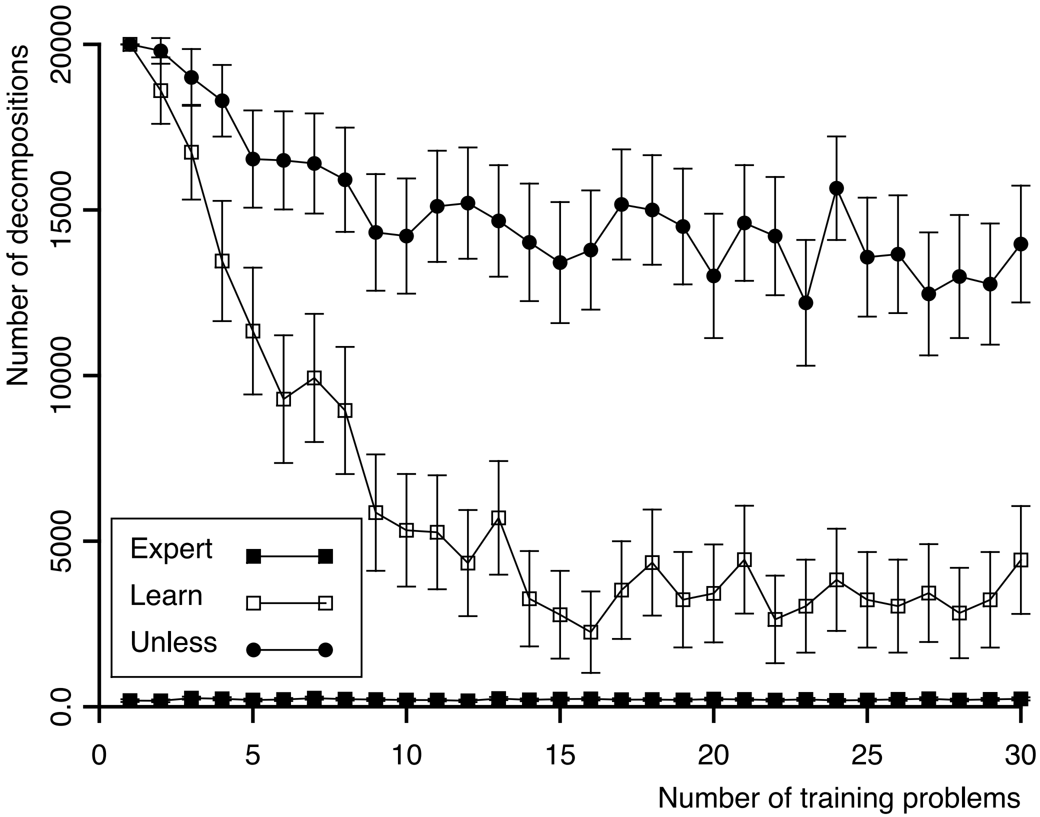
*But expertise relies **less** on goal conditions.*

Each curve shows means with 95% confidence intervals over 100 orders.

HPNL Results on Depots

Number of decompositions needed to solve problems in the Depots domain.

Learning curves are similar in this domain.



But HPNL never defeats search completely.

Each curve shows means with 95% confidence intervals over 100 orders.

Related Research

Our approach to learning plan knowledge shares some themes with previous work:

- Acquiring HTNs/HGNs from sample plans / solutions
  - Ilghami et al. (2002), Hogg et al. (2008), Nejati et al. (2006)
- Learning decomposition rules from problem solving
  - Marsella (1993), Shavlik (1990), Reddy / Tadepalli (1997)
- Inductive programming (Schmid & Kitzelmann, 2011)
- Meta-interpretive learning (Cropper & Muggleton, 2015)

However, there are important differences, such as our use of domain constraints and goal interactions.

Plans for Future Work

In future research, we will carry out more experiments that:

- *Compare our approach to classic EBL and ILP techniques*
- *Examine other planning domains to ensure generality*

We will also improve HPNL's learning rate by replacing:

- *Substitution of constants with variables by propagation of dependencies through sample plans*
- *Chaining technique for static relations with specialized form of inductive logic programming*

These should produce HPN methods that generalize better to novel test problems.

Summary Remarks

This talk reviewed hierarchical problem networks and their use for knowledge-based planning. It also:

- Specified the task of learning HPNs from sample plans
- Presented a novel approach to this task that:
  - *Maps each plan decomposition onto an HPN method*
  - *Relies on domain constraints to find state conditions*
  - *Examines goal interactions to identify goal conditions*

Experiments with three domains suggest that the approach learns effective HPNs very rapidly.

References

Cropper, A., & Muggleton, S. H. (2015). Learning efficient logical robot strategies involving composable objects. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence* (pp. 3423–3429). Buenos Aires: AAAI Press.

Hogg, C., Muñoz-Avila, H., & Aha, D. W. (2008). HTN-Maker: Learning HTNs with minimal additional knowledge engineering required. *Proceedings of the Twenty-Third National Conference on Artificial Intelligence*. Chicago: AAAI Press.

Ilghami, O., Nau, D. S., Muñoz-Avila, H., & Aha, D. W. (2002). CaMeL: Learning method preconditions for HTN planning. *Proceedings of the Sixth International Conference on AI Planning and Scheduling* (pp. 131–141). Toulouse: AAAI Press.

Nejati, N., Langley, P., & Könik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the Twenty-Third International Conference on Machine Learning* (pp. 665–672). Pittsburgh, PA.

Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 278–286). Nashville, TN: Morgan Kaufmann.

Schmid, U., & Kitzelmann, E. (2011). Inductive rule learning on the knowledge level. *Cognitive Systems Research*, *12*, 237–248.

Shavlik, J. (1990) Acquiring recursive and iterative concepts with explanation-based learning. *Machine Learning*, *5*, 39–70.

Comparison to Alternatives

Hierarchical problem networks share features with earlier planning frameworks, but also have important differences.

REPRESENTATIONAL AND PROCESSING ASSUMPTIONS	*Classic Planners*	*HTN Planners*	*HGN Planners*	*HPN Planners*
Generate sequential plans that achieve goals	●	●	●	●
Decompose complex activities hierarchically	○	●	●	●
Methods require that relations hold in state	○	●	●	●
Methods indexed by goals they achieve	○	○	●	●
Decompose problems into subproblems	○	○	○	●
Methods require that goals are not unsatisfied	○	○	○	●
Methods are linked to primitive operators	○	○	○	●

This table compares HPNs with HTNS, HGNs, and classic planners along seven dimensions.