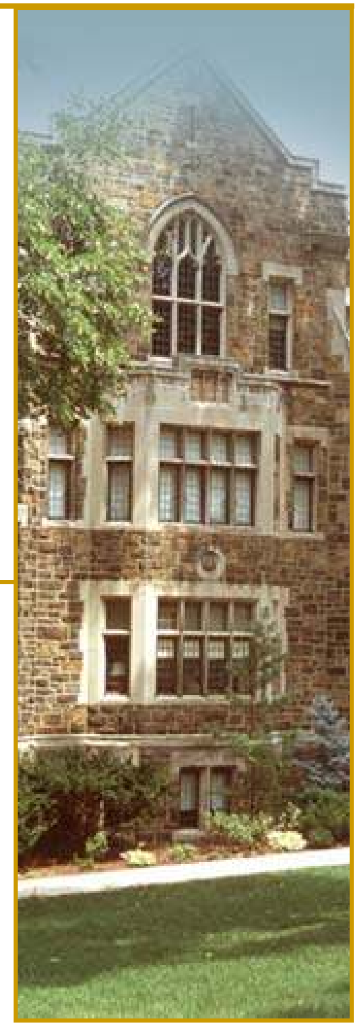
LEHIGH University

Learning of Hierarchical Task Network Domain Descriptions: Theory and Empirical Results.



Héctor Muñoz-Avila
Dept. of Computer Science & Engineering
Lehigh University

Chad Hogg
Ke Xu

Okthay Ilghami
Ugur Kuter

Outline

- Lehigh University
  - The InSyTe Laboratory
- Motivation: learning hierarchies
- Background
  - Hierarchical Task Network (HTN) planning
  - Problem description
- Learning structure of HTNs
- Learning preconditions of HTNs
- Final remarks

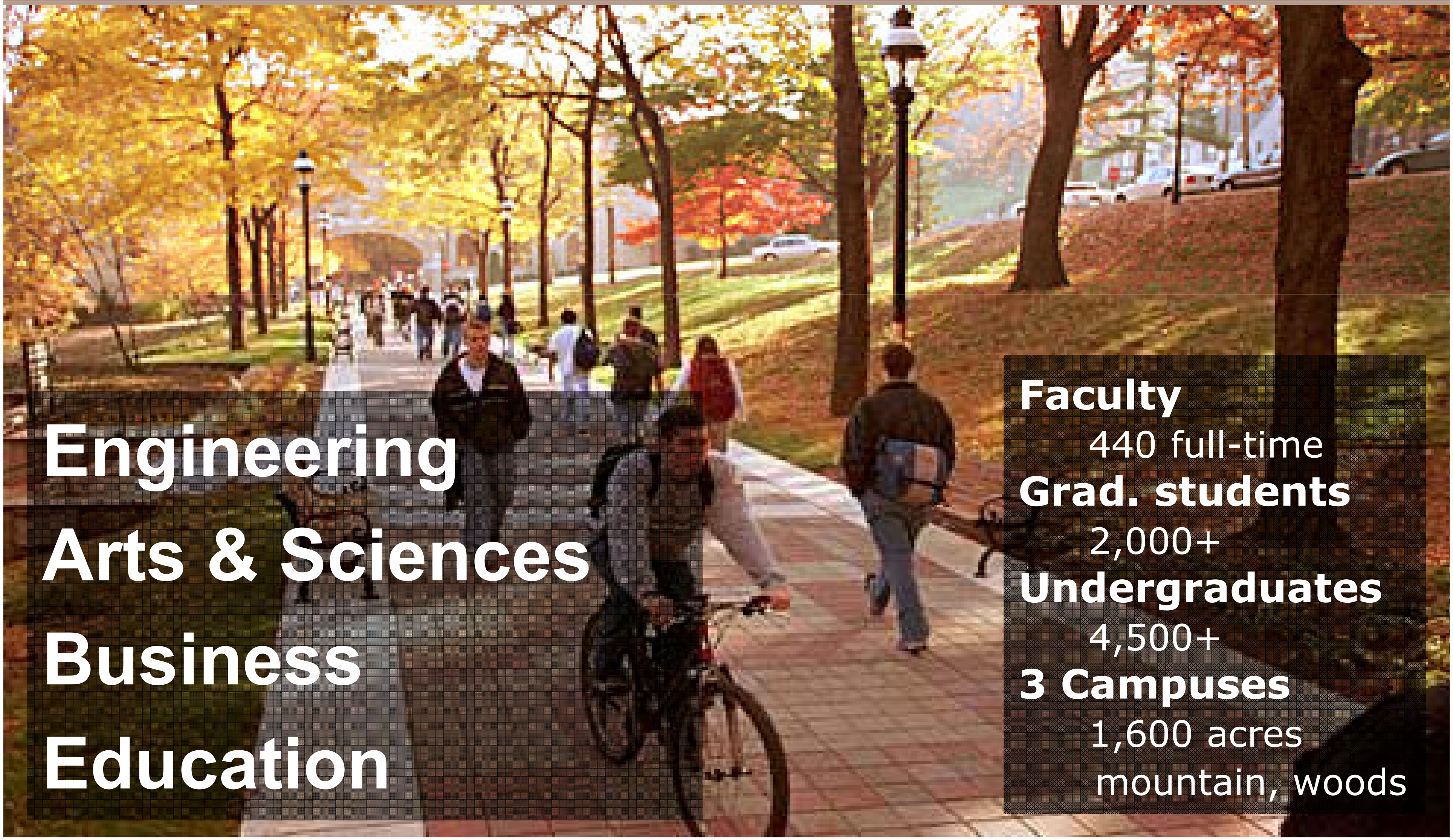
LEHIGH University

LEHIGH

UNIVERSITY

LEHIGH UNIVERSITY

The University



Engineering

Arts & Sciences

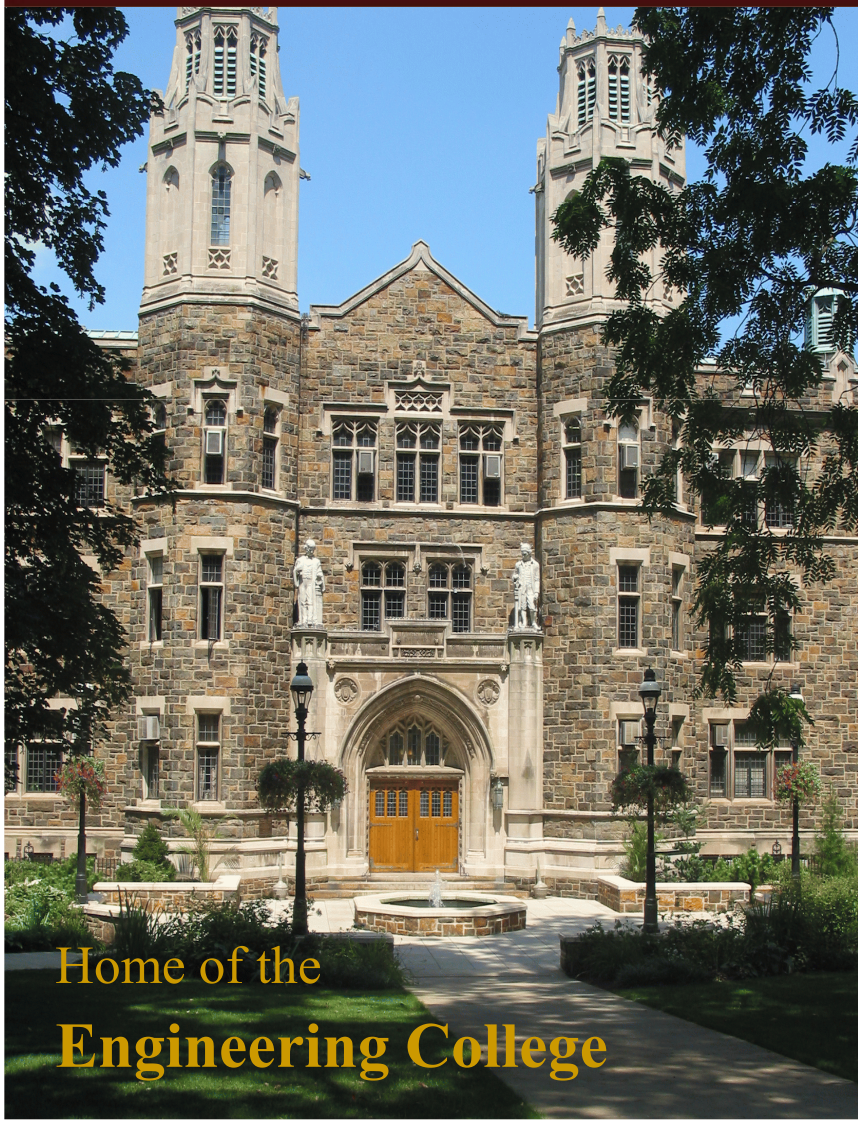
Business

Education

Faculty
440 full-time
Grad. students
2,000+
Undergraduates
4,500+
3 Campuses
1,600 acres
mountain, woods

LEHIGH UNIVERSITY

Computer Science & Engineering



- Ph.D. and Masters programs
  - Computer Science
  - Computer Engineering
- Faculty
  - 16 tenured / tenure-track faculty
- Graduate Students
  - >35+ PhD students
  - >35+ MS students

Engineering College

top 20% of US PhD Engr schools

University

top 15% of US National Univs.

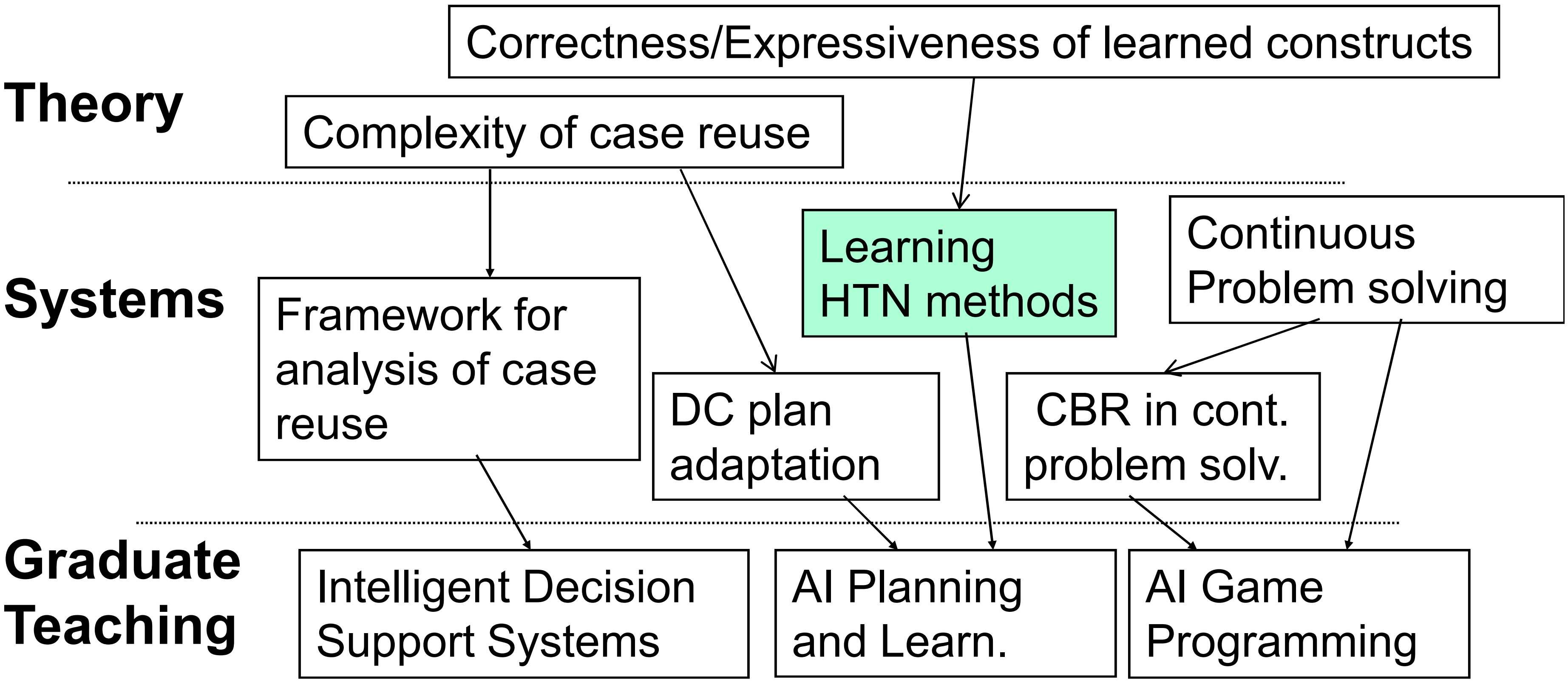
LEHIGH UNIVERSITY

CSE Research Areas

- Artificial intelligence
- Bioinformatics
- Computer architecture
- Database systems
- Enterprise information systems
- Electronic voting
- Game AI
- Graphics
- Networking
- Pattern recognition & computer vision
- Robotics
- Semantic web
- Software engineering
- Web search

InSyTe Lab

- intersection of case-based reasoning, planning, and machine learning



Motivation

Lehigh University

Traveling to Hong Kong University

Traveling to airport …

Getting to the car

Walk to door

Open door

…

Driving to airport

Driving local route

Drive car 3rd street

Driving highway

Drive route 22

…

Park at Lehigh airport

…

Hong Kong University

General problem: Can we create algorithms capable of extracting such hierarchies automatically?

Why learning such hierarchies?

Traveling to Hong Kong University

Traveling to airport …

Getting to the car

Walk to door

Driving to airport

ng local route

Driving highway

Lehigh airport

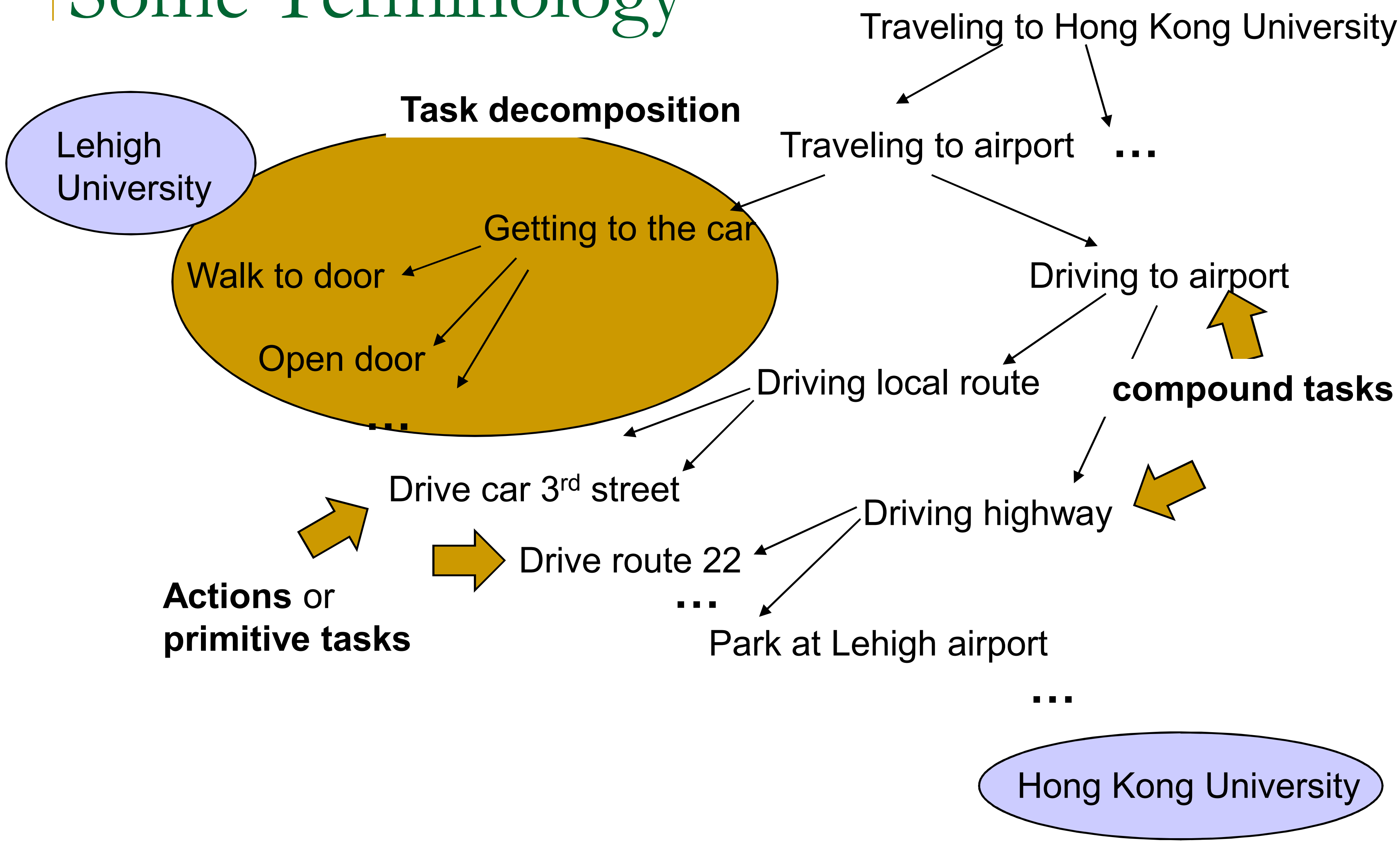
…

Lehigh University

Hong Kong University

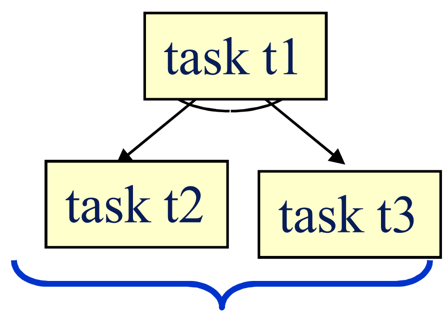
- Automated recognition
  - ➢ So we can learn the purpose/**objective** of an agent
- Learning **strategies**
  - ➢ So these strategies can be **reused** to achieve these objectives in different situations

Some Terminology



Hierarchical Task Network (HTN) Planning

- Complex tasks are decomposed into simpler tasks.



- Seeks to decompose ***compound*** tasks into ***primitive*** tasks, which define actions changing the world
- Primitive tasks are accomplished by knowledge artifacts called **operators**
- The knowledge artifacts indicating how to decompose task into subtasks are called ***methods***

Basic HTN Knowledge Constructs

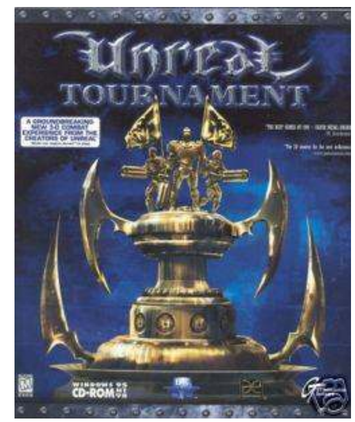
- Methods: Indicate how to decompose a compound task
  - **Task**: drive-to-airport ?p ?c ?a
  - **Preconditions**: person ?p, airport ?a, car ?c, in ?p ?c, location ?l1, location ?l2, at ?c ?l1
  - **Subtasks**: drive-local-road ?c ?p ?l1 ?l2,
    drive-highway ?c ?p ?l2 ?a

- Operators: indicate how to execute a primitive task
  - **Task**: drive ?c ?l1 ?l2
  - **Preconditions**: car ?c, at ?c ?l1, ?l1 ≠ ?l2
  - **Effects**: at ?c ?l2, ¬ (at ?c ?l1)

LEHIGH University

Why HTN Planning?

- HTN planning has been shown to be more expressive than classical plan representations (Erol et al, 1994)
  - Using methods and operators versus using operators only
- It is natural in many real-world applications
  - e.g., modeling strategies in computer games



- Fast planning through domain-configurable HTN planners (SHOP system)

Annotated Tasks

- Tasks in HTN planning are simply atomic symbols
  - e.g., travel ?p ?L

- For the purposes of our learning algorithm we introduce annotated tasks:
  - A **task description** indicates its preconditions and effects
    - **Task**: travel ?p ?l
    - **Preconditions**: person ?p, location ?l
    - **Effects**: at ?p ?l

- Annotated tasks are used in other areas including process models

Concrete Learning Problem

- **Given**:
  - A collection of plans – sequences of actions
    - e.g., plan getting from Lehigh to HKUST
  - A collection of task descriptions
  - A collection of operators
- **Obtain**:
  - A collection of methods for accomplishing the tasks

HTN-MAKER

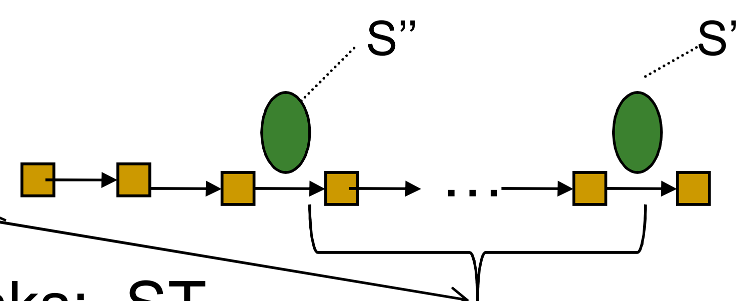
- Solves the learning problem stated in the previous slide
- It does so in an **incrementally**
- It does so in a **sound** way
  - Methods learned are such that any plan generated by an HTN planner for a task is consistent with the task description
- It is conditionally **complete**
  - There is a finite collection of (problem, solution plan) pairs such that when fed to HTN-MAKER yields a complete domain relative to a fixed set of input task descriptions
- It is **expressive**
  - Methods learned can be used to represent problems that are not representable as STRIPS (e.g., action-based) problems

HTN-MAKER: Basic Steps

Input: plan π, state S, task description T

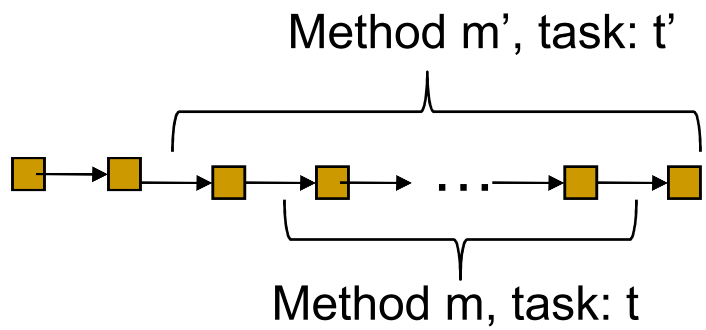
1. S’ ← S; A ← first action in π
2. Select a task t ∈ T such that:
   - effects of t are satisfied in S’, and
   - preconditions of t are satisfied in a state S’’ preceding S’ wrt π
3. p ← regressConditions(S’’,S’, π)
4. ST ← collectActions(S’’,S’, π)
5. Construct method:

   task: t, preconditions: p, subtasks: ST
6. S’ ← apply(A,S’); A ← next-action(A, π)
7. Go back to 2 until A = null



HTN Maker: Further Considerations (1)

- Hierarchies appear naturally when methods are subsumed by other methods:



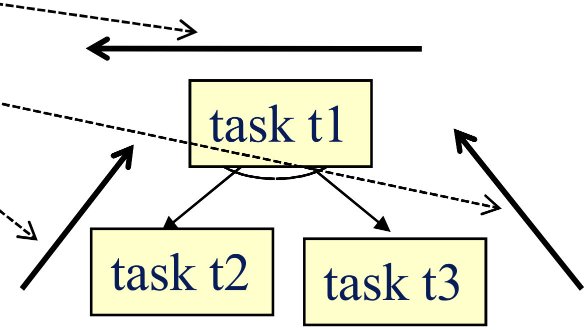
task t will appear as a subtask of t’ in method m’

  - Special case: if t = t’ then will learn recursive methods

- Multiple tasks can be selected: choice must be made
  - How to group tasks: left-recursive, right-recursive, other?
    - Currently right-recursive

HTN Maker: Further Considerations (2)

- Initial algorithm found not to be sound
  - Need to add ***verifier task*** as last subtask for every method achieving a task *t*. Verifier tasks are achieved by a new method:
    - Preconditions: the effects of *t*
    - Subtasks: none
- Regressing conditions at higher levels of the hierarchy
  - Pushing conditions from lower levels
  - *Horizontal* and *vertical* goal regression



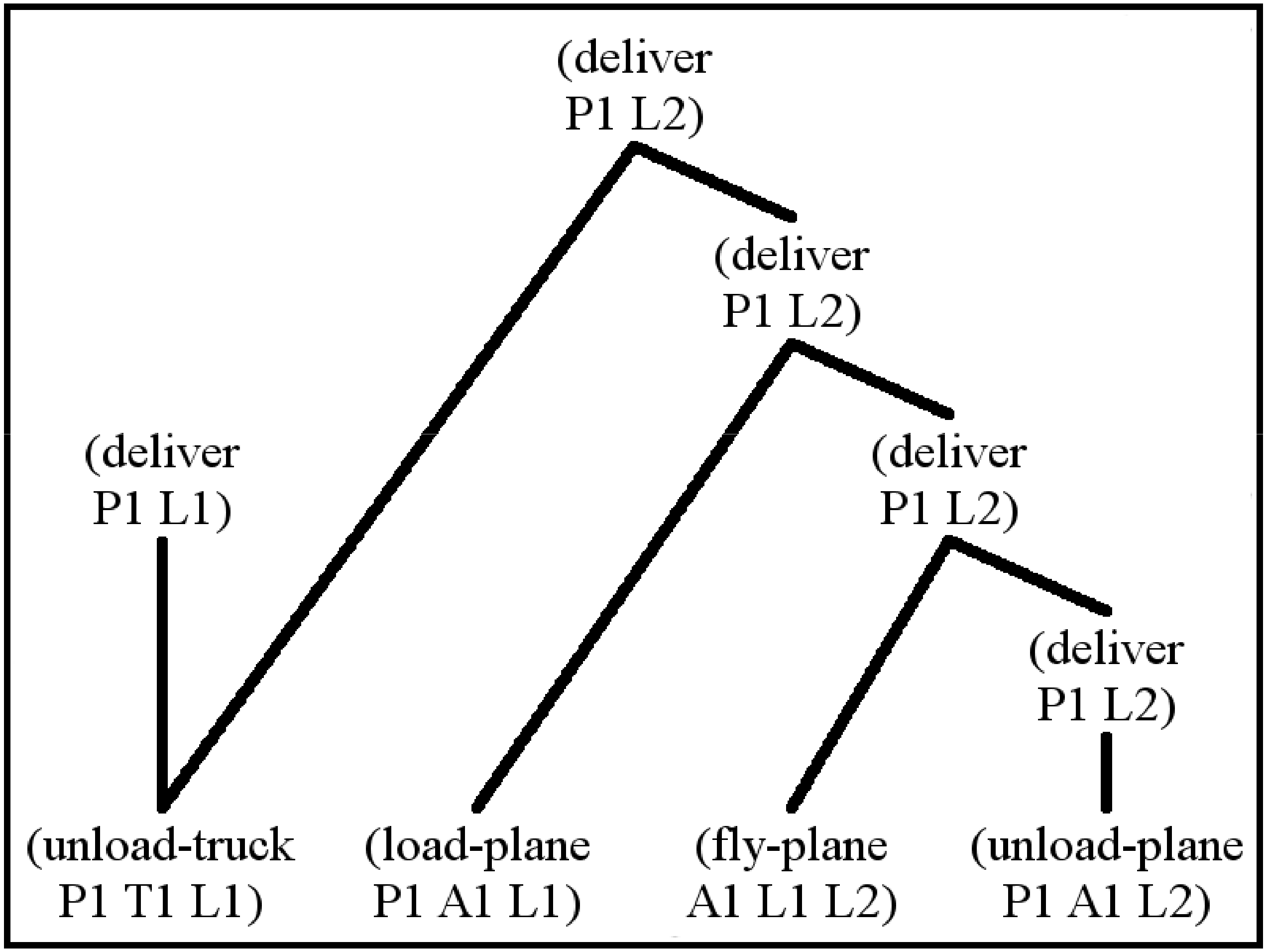
- Detecting opportunities to avoid learning unnecessary methods

Example

- Domain:
  - Logistics

- Initial state:
  - in P1 T1
  - at T1 L1
  - at A1 L1
  - …

- Single task:
  (deliver ?p ?l)
  - precond.: none
  - effect: (at ?p ?l)

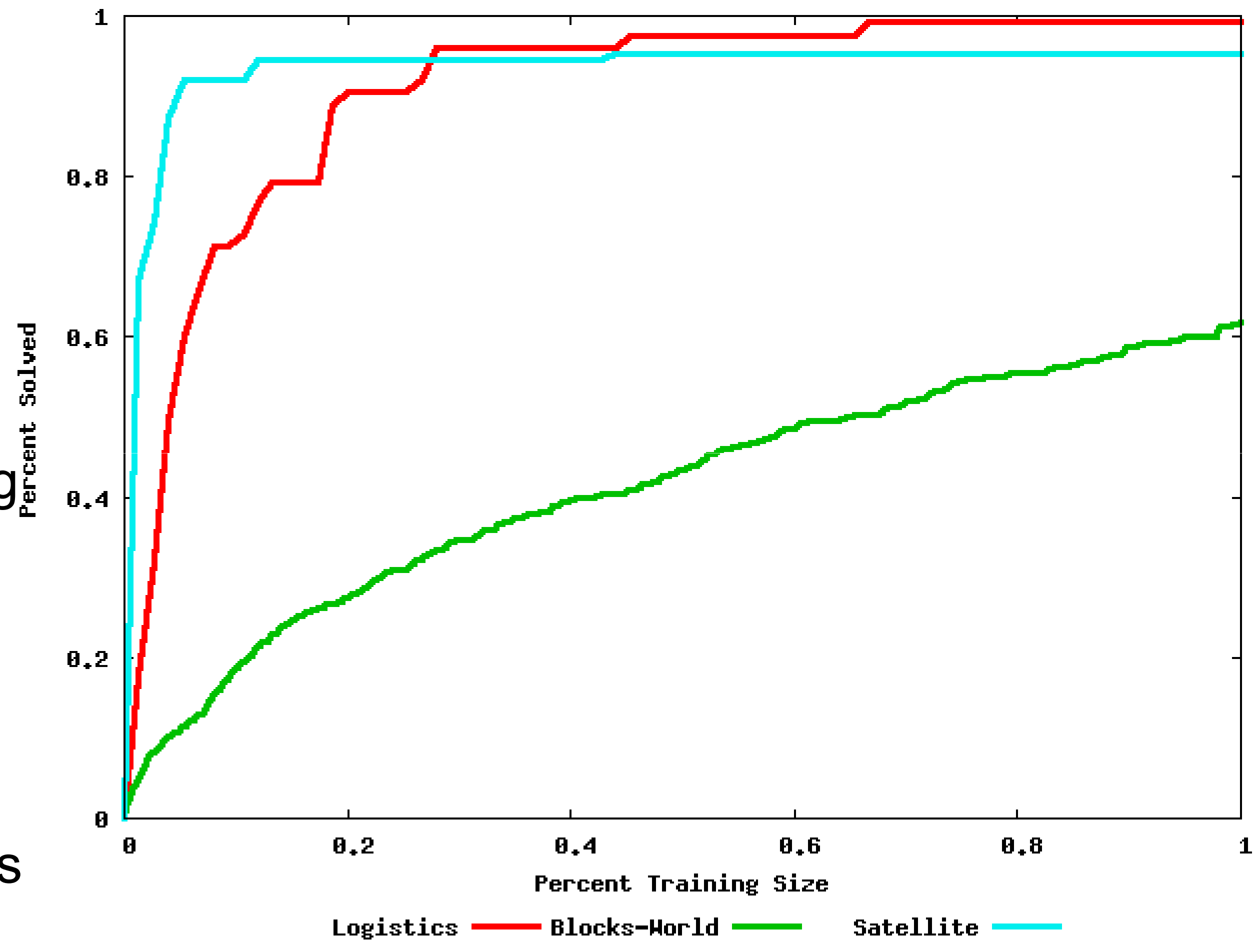


Empirical Evaluation

- Domains:
  - ➢ Logistics
  - ➢ Blocks World
  - ➢ Satellite

- Testing set:
  - ➢ 20% of training set size
  - ➢ Independently generated from training set

- Average over 5 runs



Learning Preconditions

- **Problem**
  - **Input**:
    - A collection of plans and the HTNs entailing them
    - An action model
  - **Output**: the preconditions for the task decompositions
- **We explored two alternatives:**
  - Using Inductive Logic Programming methods (ILP)
    - Might yield incorrect generalizations
    - Requires additional **input**: ontology of objects
    - Does not require convergence
  - Using Version Spaces
    - Always yield correct generalizations
    - Requires additional **input**: negative examples
    - Requires sufficient examples for full convergence

LEHIGH University

First Alternative: Generalization (ILP)

Concrete Decomposition

Head (Task):

Transport $Package_{100}$ Bethlehem Pittsburgh

Conditions :

City Bethlehem

City Pittsburgh

City $Package_{100}$

Truck $Truck_{47}$

SubTasks:

Drive $Truck_{47}$ Bethlehem

Load $Truck_{47}$ $Package_{100}$

Drive $Truck_{47}$ Bethlehem Pittsburgh

Unload $Truck_{47}$

Method

Head (Task):

Transport $?pkg_1$ $?ct_1$ $?ct_2$

Preconditions:

City $?ct_1$

City $?ct_2$

Package $?pkg_1$

Truck $?truck_1$

SubTasks:

Drive $?truck_1$ $?ct_1$

Load $?truck_1$ $?pkg_1$

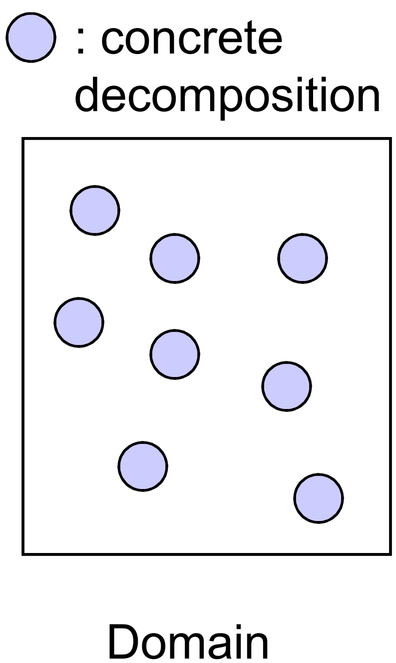
Drive $?truck_1$ $?ct_1$ $?ct_2$

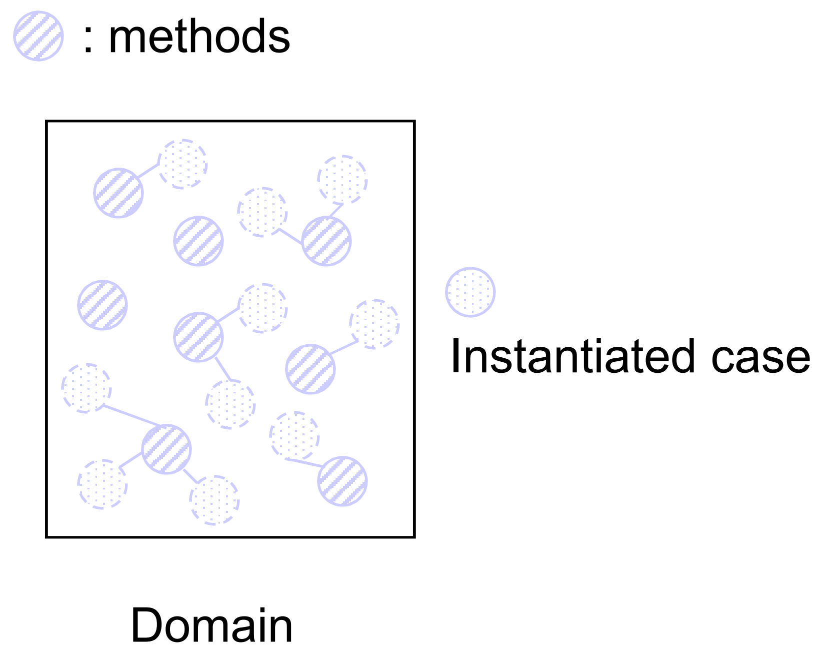
Unload $?truck_1$

Generalization gives Better Coverage

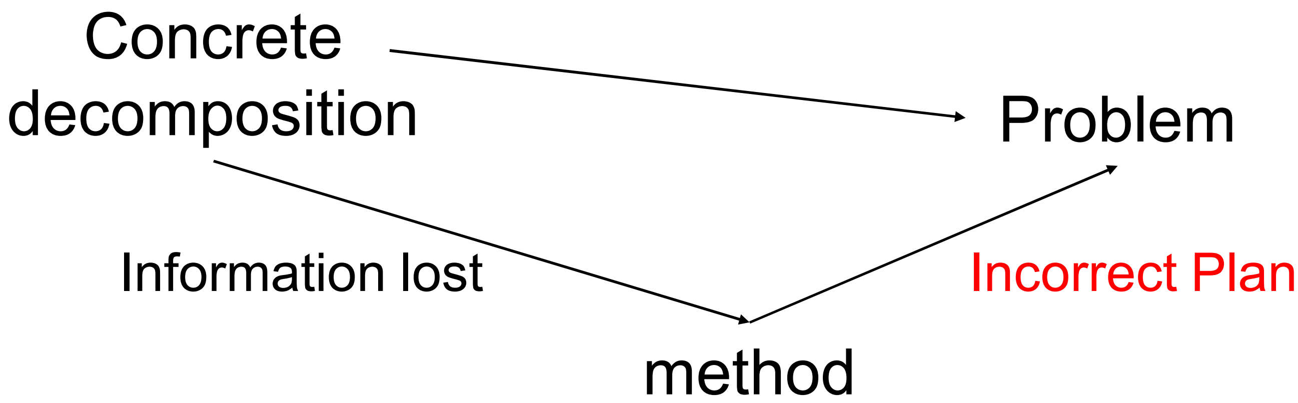
Coverage(CB) = {*p*: *p* is a planning problem that can be solved by using a knowledge base *CB*}

(Smyth & Keane, 1995)





Problem: Over-generalization



We will annotate methods with the following information:

- Original bindings
- Type Ontology: A collection of relations between objects
  - Examples:
    - ?X **type:** V (?x **type:** Tanker)
    - V’ **isa** V (Tanker **isa** Vehicle)

Method Over-generalization (cont.)

Method M_1

Task:

deliver $?e_1$? ...

Conditions:

$?t_5$ *type* Tanker

$?e_1$ *type* Liquid

…

Both Applicable

Type Ontology

Method M_2

Task:

... $?d_6$ $?d_8$

Conditions:

$?t_2$ *type* RefrigTanker

$?e_4$ *type* PerishableLiquid

…

Problem:

Task: deliver Milk from Location_X Location_Y

State: Type Tanker_A Tanker ✗ M_1

Type Tanker_B RefrigTanker ✓ M_2

Type Milk PerishableLiquid

Solution: Preference-Guided Retrieval

- General Idea
  - Detect conflicting conditions between methods to elicit preferences automatically
  - Use preferences to guide method retrieval (**Case-based reasoning** approach)
- Phase 1 – Method Refinement
  - Type Preferences: reduce method over-generalization
  - Constant Preferences: preserve original variable bindings
- Phase 2 – Method Retrieval
  - Rank refined cases with a similarity criterion
  - Bias: prefer specificity over generality

Input decomposition → [Method Refined] → Preferences → [Method Ranked] → Retrieved method

Case Refinement: Preference Elicitation

Method M_1	Method M_2
Task:	**Task**:
deliver $?e_1$ $?d_7$ $?d_3$	deliver $?e_4$ $?d_6$ $?d_8$
Conditions:	**Conditions**:
$?t_5$ *type* Tanker	$?t_2$ *type* RefrigTanker
$?e_1$ *type* Liquid	$?e_4$ *type* PerishableLiquid
Preferences:	**Preferences:**
Constant Preferences	Constant Preferences
equal $?e_1$ e_1	equal $?e_4$ e_4
equal $?d_7$ d_7	equal $?d_6$ d_6
equal $?d_3$ d_3	equal $?d_8$ d_8
equal $?t_5$ t_5	equal $?t_2$ t_2
Type Preferences	
not $?t_5$ type RefrigTanker	
not $?e_1$ type PerishableLiquid	

Method Retrieval: Preference-Based Similarity

Method

Knowledge base

New problem

$$sim(gC, CB, P) = appl * (w_1 * sc + w_2 * stp + w_3 * scp)$$

applicability value (1 or 0)

conditions / MAX # conditions

satisfied type preferences / # preferences

satisfied constant preferences / # preferences

Reduce Case Over-generalization

Method M_1	Method M_2
Not Retrieved	Retrieved with Higher Similarity
$?t_5$ *type* Tanker	$?t_2$ *type* RefrigTanker
$?e_1$ *type* Liquid	$?e_4$ *type* PerishableLiquid
Preferences Added	Preferences Added

Problem:

Task: deliver Milk from Location_X Location_Y

State: Type Tanker_A Tanker

Type Tanker_B RefrigTanker

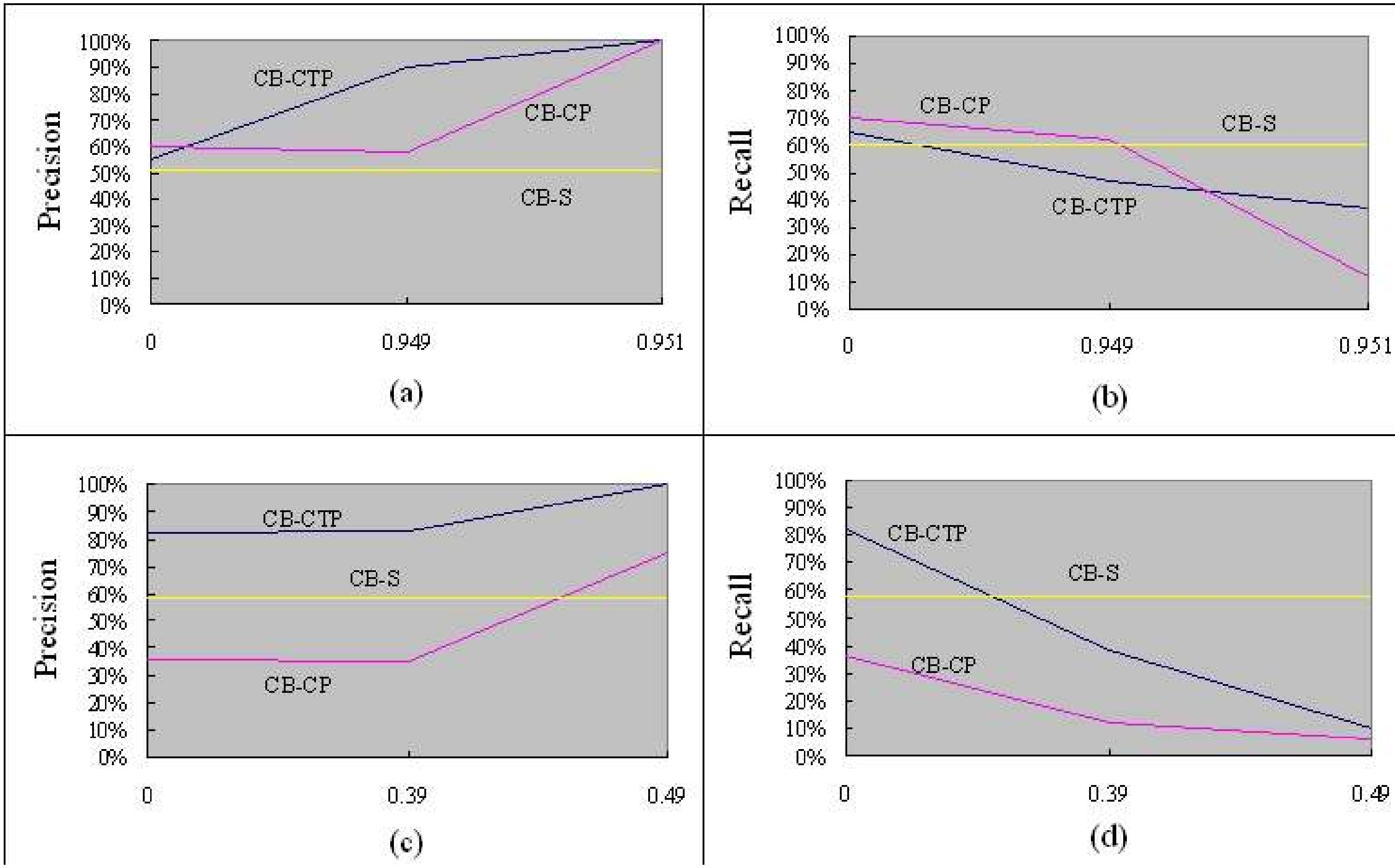
Type Milk PerishableLiquid

Properties

- *PS*: The set of the input problem-solution episodes (the training set)
- *CB-C*: concrete methods generated from *PS*
- *CB-S*: methods obtained from generalization of *CB-C*
- *CB-CP*: methods adding constant preferences to *CB-S*
- ***CB-CTP***: methos adding type preferences to *CB-CP*

- ***<u>Properties</u>***:
- *CB-C*, *CB-CP*, and ***CB-CTP*** are sound relative to *PS*
- *CB-S* is not sound relative to *PS*
- *CB-S*, *CB-CP*, and ***CB-CTP*** have the same coverage
- *CB-C* has less coverage than ***CB-CTP***

Empirical Results



UMTranslog Domain

Process Planning Domain

X-axis: similarity threshold

Second Alternative: Use Version Spaces

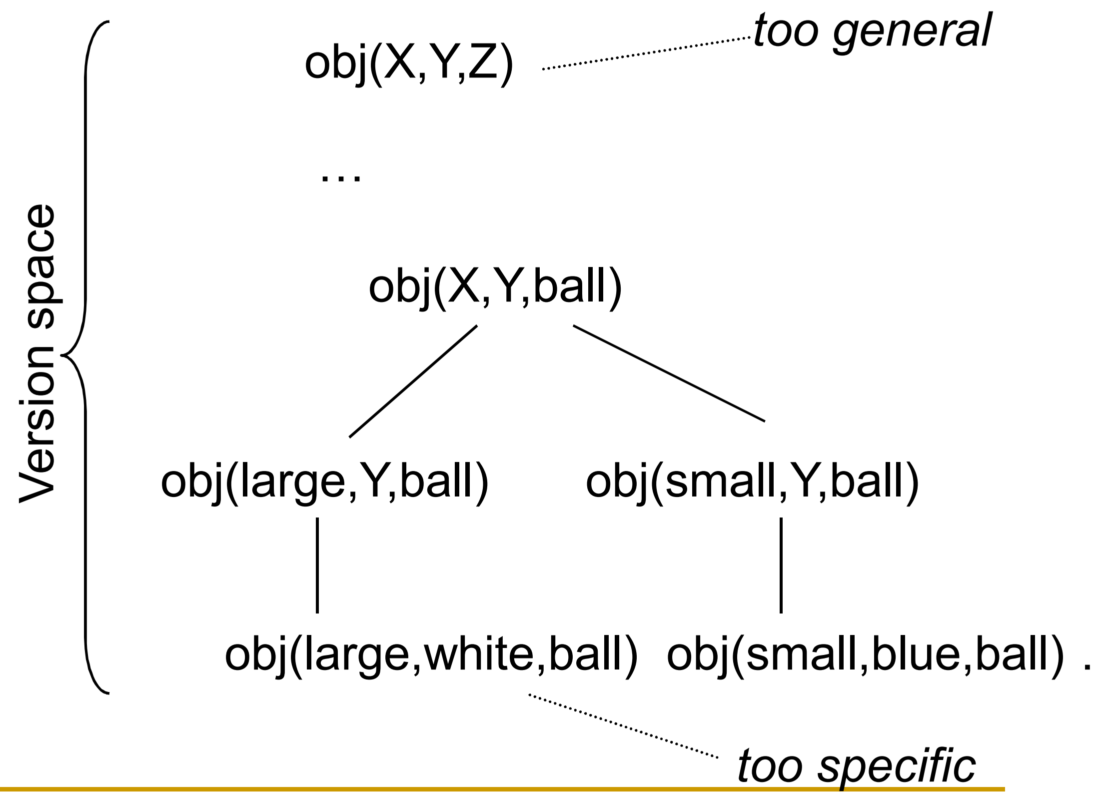
Idea: Learn a concept from a group of instances, some positive and some negative

Example:

- target: obj(Size,Color,Shape)
  - Size = {large, small}
  - Color = {red, white, blue}
  - Shape = {ball, brick, cube}

- Instances:
  - +:
    - obj(large,white,ball)
    - obj(small,blue,ball)
  - −:
    - obj(small,red,brick)
    - obj(large,blue,cube)

Two extremes (temptative) solutions:



How To Use This for Learning Preconditions

- The range of each variable in a predicate is known
- Then we can do generalizations/specializations via the following operations on variables:
  - instantiation: P(?x, c) is more general than P(a,c)
  - disjunction: P((a or b), c) is more general than P(a, c)
  - negation: P((anything other than d), c) is
    - more general than P((a or b), c) and P(a, c)
    - less general than P(?x,c)
- **Normalization:**
  - Take constants that play the same role (e.g., *truck1* and *truck2*)
  - Replace them with the same variable (e.g., ?*t*)
- **Inductive bias**: concept is in hypotheses space

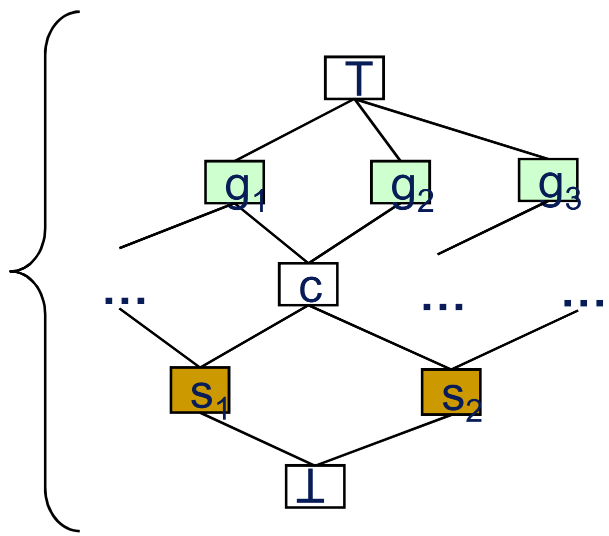
Solution: Version Spaces

- For each method, maintain a version space for its preconditions

task:
get(*p*)

subtasks:
walk(?*x*,?*y*)
take(*p*)
walk(?*y*,?*x*)

precond:



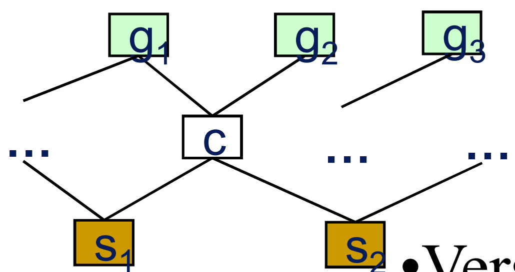
Solution: Version Spaces

- For each method, maintain a version space for its preconditions

task:
get(*p*)

subtasks:
walk(?*x*,?*y*)
take(*p*)
walk(?*y*,?*x*)

precond:



•Update the version space, using Candidate Elimination each time that the method is used in a plan trace

•Version spaces from other methods may need to be updated as well (Ilgami *et al*., 2003)

Solution: Version Spaces

- For each method, maintain a version space for its preconditions

task:
get(*p*)

subtasks:
walk(?*x*,?*y*)
take(*p*)
walk(?*y*,?*x*)

precond:

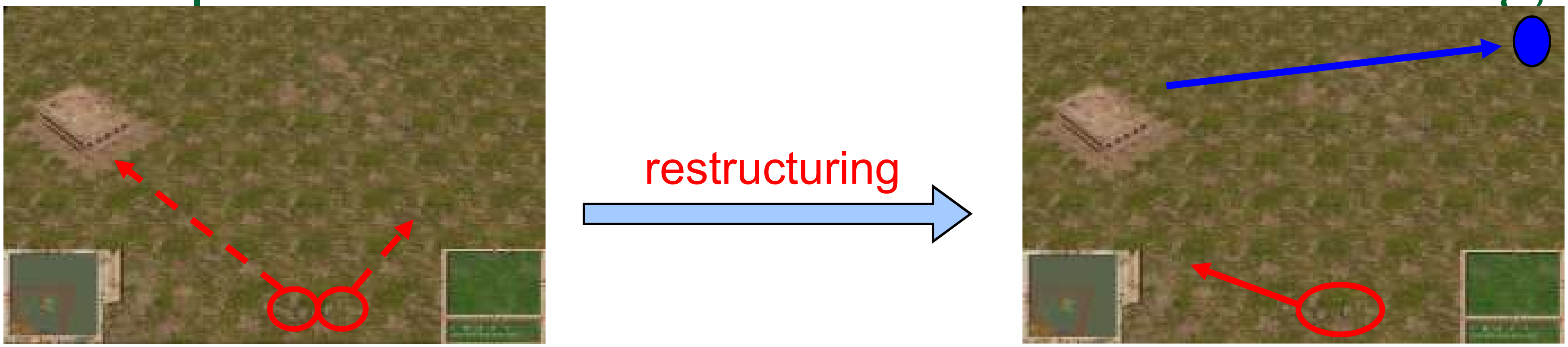
c

- Update the version space, using Candidate Elimination each time that the method is used in a plan trace

- Version spaces from other methods may need to be updated as well (Ilghami *et al*., 2003)

- Terminate when version space converges to a single hypothesis

Empirical Validation: Transfer Learning



Task A: Learning to control a specific location on a map

Task B: Learning to control another location on a map

Transferred knowledge:

- Conditions for applying an operational procedure (represented as an HTN)

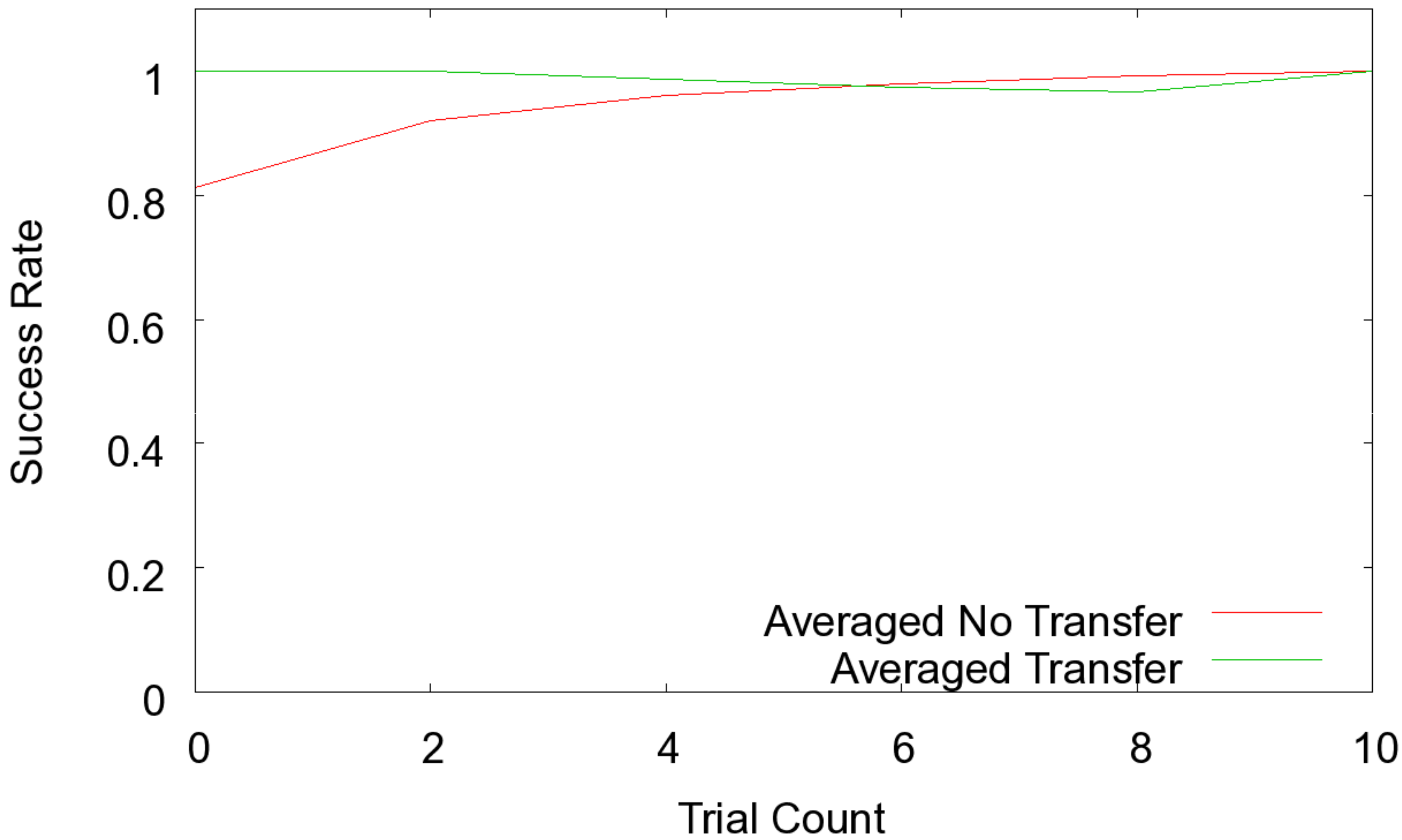
Performance Goal:

- Frequency with which a soldier unit learns to control a location on a map

Background Knowledge Used:

- Operational procedure of how to control a location (HTNs without conditions)

Empirical Results



Final Remarks

- We presented HTN-MAKER an algorithm capable of learning sound and complete domain descriptions for HTN planning
- System demonstrated convergence in 3 domains used in the IPC
- We also presented alternative approaches to task decomposition
  - One based on ILP
  - Another one based on version spaces
- Future work:
  - CPU performance versus STRIPS planning?
  - Integration of ILP-based approach in HTN-MAKER
  - **Currently**: using reinforcement learning in HTN-MAKER for domains with uncertainty

Bibliography

- Hogg, C., Muñoz-Avila, H., and Kuter, U.. *(2008)* HTN-MAKER: Learning HTNs with Minimal Additional Knowledge Engineering Required. *To appear in Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08).*
- Hogg, C. & Muñoz-Avila, H. (2007) Learning of Tasks Models for HTN Planning. *Proceedings of the ICAPS-07 Workshop on AI Planning and Learning (AIPL)*. AAAI Press.
- Xu, K. & Muñoz-Avila, H. (2007) CaBMA: A Case-Based Reasoning System for Capturing, Refining and Reusing Project Plans. *Knowledge and Information Systems (KAIS)*. Springer.
- Lee-Urban, S., Parker, A., Kuter, U., Muñoz-Avila, H., & Nau, D. (2007) Transfer Learning of Hierarchical Task-Network Planning Methods in a Real-Time Strategy Game. *Proceedings of the ICAPS-07 Workshop on ICAPS 2007 Workshop on Planning and Learning (AIPL)*. AAAI Press
- Ilghami, O., Nau, D.S., and Muñoz-Avila, H. (2006) Learning to Do HTN Planning. *Proceedings of the International Conference on Automated Planning & Scheduling (ICAPS-06)*. AAAI Press.
- Xu, K and Muñoz-Avila, H. (2005) A Domain-Independent System for Case-Based Task Decomposition without Domain Theories. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*. AAAI Press.
- Ilghami, O., Muñoz-Avila, H., Nau, D.S., and Aha, D. Learning Approximate Preconditions for Methods in Hierarchical Plans. *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)*. AAAI Press.

Questions?