A Unified Framework for Knowledge-Lean and Knowledge-Rich Planning

Son Thanh To

Pat Langley

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

Dongkyu Choi

Department of Aerospace Engineering
University of Kansas, Lawrence, KS

This research was supported by Grant N00014-10-1-0487. We thank M. Barley, U. Kuter, S. Kambhampati, H. Muñoz-Avila, D. Nau, and M. Stilman for useful discussions.

Two Paradigms for Planning

The ability to create novel plans is a distinctive characteristic of human cognition, utilizing:

- Extensive search on unfamiliar tasks (Newell & Simon, 1972)
- Routine activity when they have expertise (Larkin et al., 1980)

Humans can also switch between these two modes and even interleave them as appropriate.

In contrast, AI has developed two paradigms – *first-principles planning* and *HTN planning* – that are almost entirely disjoint.

Cognitive systems should aim to unify these two frameworks.

Knowledge-Lean Planning

The first-principles paradigm defines a planning problem as:

A set of literals S that describe an initial state

A goal formula G that describes a class of goal states

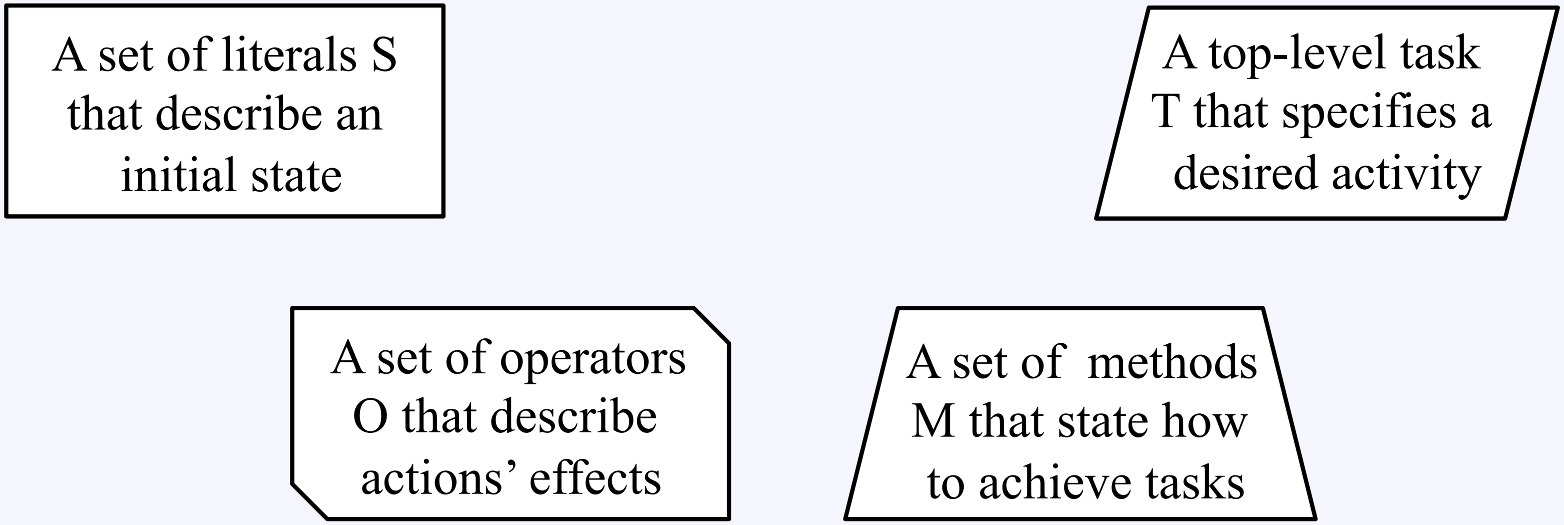
A set of operators O that describe actions' effects

Techniques for solving such tasks (e.g., backward or forward chaining) rely on extensive search.

Even with heuristic guidance, *these schemes scale poorly* with the number of objects, branching factor, and solution length.

Knowledge-Rich Planning

The hierarchical task network paradigm defines a problem as:



A set of literals S that describe an initial state

A top-level task T that specifies a desired activity

A set of operators O that describe actions’ effects

A set of methods M that state how to achieve tasks

Techniques for HTN planning need little search and scale well.

But *they require a developer to engineer their knowledge,* which is time consuming and may introduce error*s*.

A Unified Planning Framework

In response, we have developed a theoretical framework that unifies the two paradigms by:

- Stating problems using goals, tasks, or their mixture
- Supporting planning with operators, methods, or both
- Combining state-space search with method decomposition

The new framework inherits the best features of the traditional ones while mitigating their weaknesses.

Representational Assumptions

The new unified framework defines a planning problem as:

A set of literals S that describe an initial state

A goal formula G that describes a class of goal states

A top-level task T that specifies a desired activity

A set of operators O that describe actions' effects

A set of methods M that state how to achieve tasks

Thus, both first-principles and HTN planning are special cases of the framework.

Representation: An Example

Methods

```
(travel_by_plane ?x ?y)
    conditions    (at ?x) (place ?y) (not (= ?x ?y))
    effects       (at ?y)
    subtasks      (book_direct ?x ?y) (fly ?x ?y)
(travel_by_plane ?x ?y)
    conditions    (at ?x) (place ?y) (not (= ?x ?y))
    effects       (at ?y)
    subtasks      (book_via ?x ?z ?y) (fly ?x ?z) (fly ?z ?y)
```

Operators

```
(book_direct ?x ?y)
    conditions    (at ?x) (direct_flight ?x ?y) (not (flight_ready ?x ?y))
    effects       (flight_ready ?x ?y)
(book_via ?x ?z ?y)
    conditions    (at ?x) (direct_flight ?x ?z) (direct_flight ?z ?y)
                  (not (flight_ready ?x ?y))
    effects       (flight_ready ?x ?z) (flight_ready ?z ?y)
(fly ?x ?y)
    conditions    (at ?x) (flight_ready ?x ?y)
    effects       (at ?y) (not (at ?x)) (not (flight_ready ?x ?y))
(drive ?x ?y)
    conditions    (at ?x) (have_car_at ?x) (place ?y) (not (= ?x ?y))
    effects       (at ?y) (not (at ?x)) (have_car_at ?y) (not (have_car_at ?x))
```

```
                (at A) (have_car_at C) (place A)
                (place B) (direct_flight A B)
Initial State   (direct_flight B A) (place C)
                (direct_flight B C) (place D)
                (direct_flight C B)
Goal Descr.     (at D)
```

Processing Assumptions

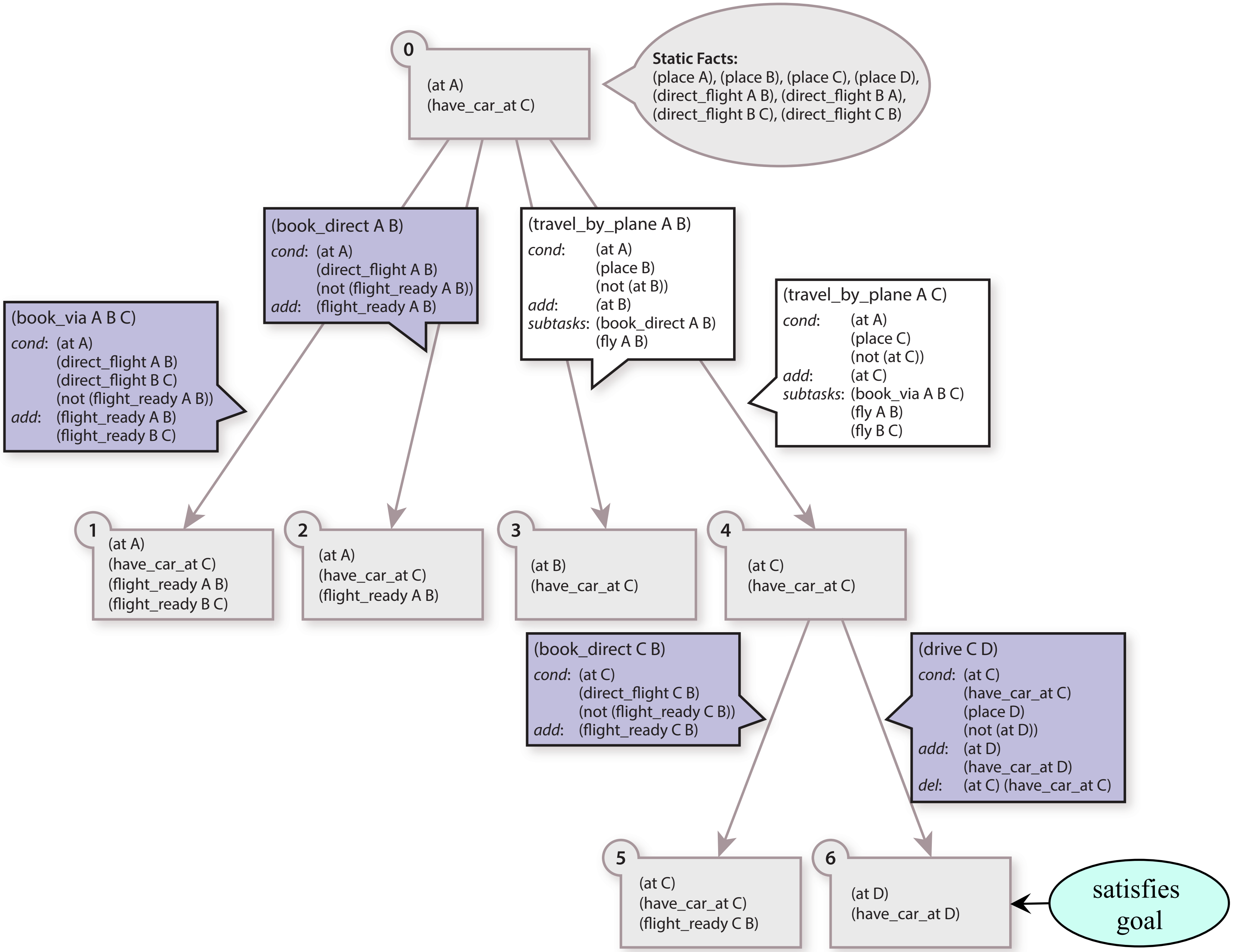
The central planning mechanism carries out forward chaining through the state space by:

- Applying a primitive operator when its conditions are met
- Expanding a nonprimitive method when its conditions match
    - Using AND/OR search to decompose it into a subplan
    - Returning the final state if it succeeds or failing otherwise
- Utilizing best-first search to organize the planning process

This approach embeds HTN-style planning within steps taken during a state-space search.

Formally, the unified framework is both sound and complete.

Processing: An Example



0

(at A)
(have_car_at C)

Static Facts:
(place A), (place B), (place C), (place D),
(direct_flight A B), (direct_flight B A),
(direct_flight B C), (direct_flight C B)

(book_direct A B)
cond: (at A)
(direct_flight A B)
(not (flight_ready A B))
add: (flight_ready A B)

(travel_by_plane A B)
cond: (at A)
(place B)
(not (at B))
add: (at B)
subtasks: (book_direct A B)
(fly A B)

(book_via A B C)
cond: (at A)
(direct_flight A B)
(direct_flight B C)
(not (flight_ready A B))
add: (flight_ready A B)
(flight_ready B C)

(travel_by_plane A C)
cond: (at A)
(place C)
(not (at C))
add: (at C)
subtasks: (book_via A B C)
(fly A B)
(fly B C)

1

(at A)
(have_car_at C)
(flight_ready A B)
(flight_ready B C)

2

(at A)
(have_car_at C)
(flight_ready A B)

3

(at B)
(have_car_at C)

4

(at C)
(have_car_at C)

(book_direct C B)
cond: (at C)
(direct_flight C B)
(not (flight_ready C B))
add: (flight_ready C B)

(drive C D)
cond: (at C)
(have_car_at C)
(place D)
(not (at D))
add: (at D)
(have_car_at D)
del: (at C) (have_car_at C)

5

(at C)
(have_car_at C)
(flight_ready C B)

6

(at D)
(have_car_at D)

satisfies goal

UPS: A Unified Planning System

We have implemented this approach in Lisp to produce *UPS*, a planner that uses a SHOP2-like notation.

The system includes three features designed to aid efficiency:

- Expanding a method instance if it selects the resulting state for expansion; if expansion fails, it abandons the branch.
- Filtering out method instances that do not produce at least one ‘useful’ state literal (e.g., that matches a goal).
- Using a numeric heuristic to guide best-first search and favor operators/methods that achieve more goals.

As in means-ends analysis, UPS uses goals to guide search, but its forward-chaining strategy does not require them.

Empirical Claims

These observations suggest four separate hypotheses about the system’s behavior:

basic ability

1. UPS solves problems stated in terms of goals, tasks, or both.
2. UPS generates plans given only operators, given a ‘complete’ set of HTN methods, or given a ‘partial’ set of methods.

search effort

3. Availability of HTN methods increases planning efficiency.
4. Filtering HTN methods for relevance further aids efficiency.

We have tested these claims on tasks of varying difficulty from Blocks World, Gripper, Logistics, Floortile, Tetris, and Visitall.

Planning with Goals and Tasks

We provided UPS with ten problems from each of the six test domains, giving it:

- Problems stated only as goal descriptions
- Problems stated only as task specifications
- Problems that included both types of structures

In each case, the system found a solution path within 150 CPU seconds that satisfied the criteria.

Knowledge Lean and Rich Planning

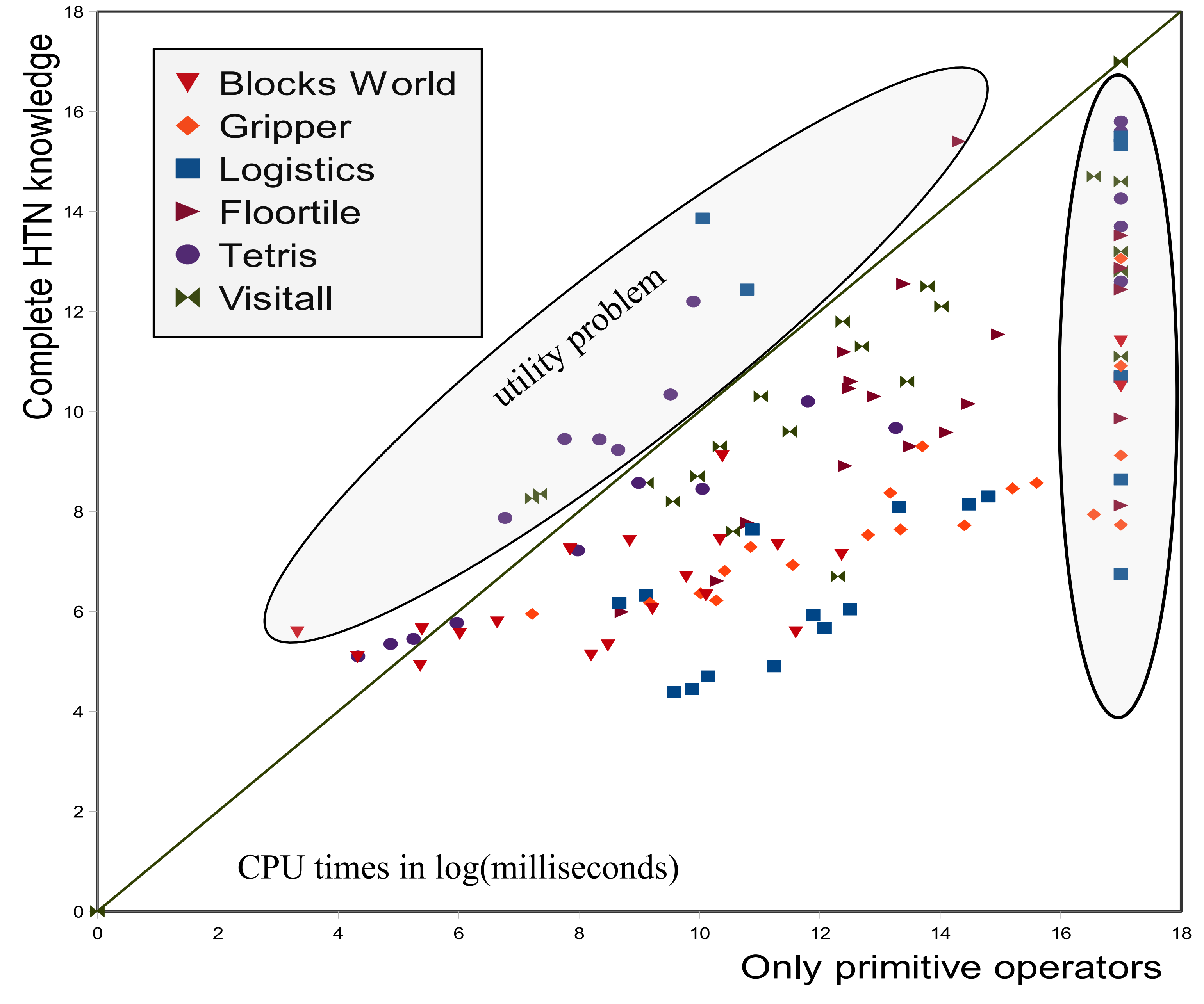
We ran UPS on ten *goal-oriented* problems from each of the six domains, giving it:

- Only a set of primitive operators
- Primitive operators and a ‘complete’ set of HTN methods
- Primitive operators and a ‘partial’ set of HTN methods

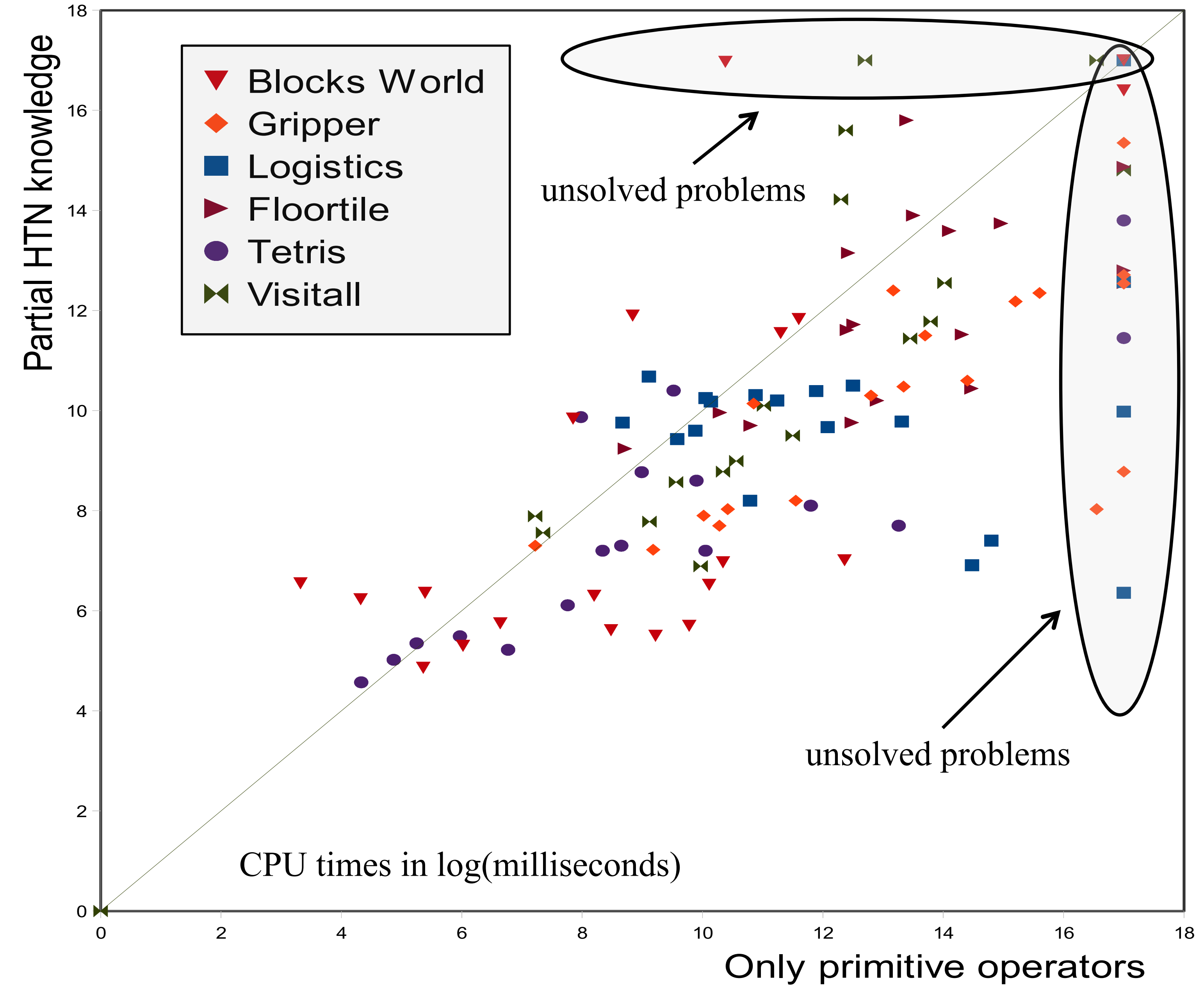
Again, the system found a solution to each problem within the allotted 150 CPU seconds.

Even without HTN knowledge, UPS’ performance was similar to entrants in the 2014 planning competition (*satisficing* track).

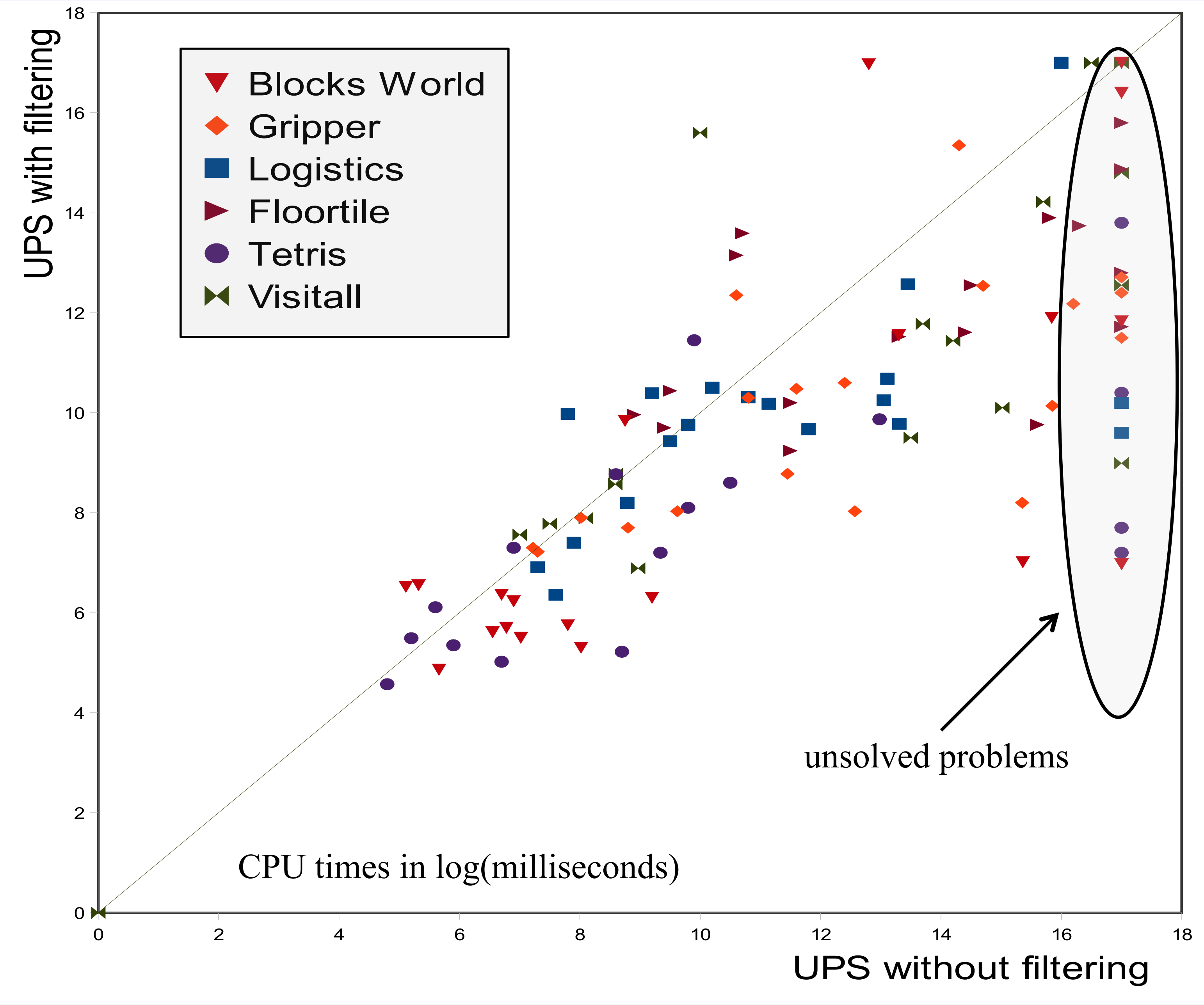
Benefits of Complete HTN Knowledge



Benefits of Partial HTN Knowledge



Benefits of Method Filtering



unsolved problems

CPU times in log(milliseconds)

Related Research

Some prior work has combined ideas from the two paradigms:

- Macro-operators (Minton, 1985; Iba, 1989; Shavlik, 1990)
  - ✦ Automatically learned, often suffer from utility problem
- Kambhampati et al.'s (1998) framework
  - ✦ Searches through a plan space, not implemented
- Gerevini et al.'s (2008) DUET system
  - ✦ Combines HTN planning with local repair, not complete
- ICARUS (Li et al., 2012)
  - ✦ Automatically learned, means-ends analysis

Our approach differs from each of these systems and offers a more natural unification of modern planning methods.

Plans for Future Work

We should explore a number of avenues in our future research:

- Clarify when knowledge aids and hinders UPS efficiency
- Incorporate these findings into improved filtering methods
- Extend UPS to support methods with quantitative effects
- Support more flexible search strategies that adapt to situation
  - E.g., forward or backward depending on branching factor
- Develop methods for learning HTN methods from solutions
  - To shift from knowledge-lean to knowledge-rich search

Together, these will provide a more complete theory and more robust approaches to planning.

Concluding Remarks

In this talk, I reviewed two paradigms for planning and offered a unified framework that:

- Handles problems stated as goals, task, or a combination
- Operates with operators, hierarchical methods, or both

Moreover, our UPS implementation of the framework:

- Solves problems without HTN methods but typically benefits from them when available
- Guides search with a goal-oriented evaluation function and filters options that appear irrelevant

Other unifications are possible, but our approach is both elegant and effective.

End of Presentation