Variations on a Theory of Problem Solving

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

Chris Pearce, Yu Bai,
Charlotte Worsfold, Mike Barley
Department of Computer Science
University of Auckland, Auckland, NZ

This research was supported by Grant N00014-15-1-2517. We thank Miranda Emery, John Laird, Rao Kambhampati, Chris MacLellan, and Manuela Veloso for useful input.

Aims of the Research

The ability to solve novel problems is a distinctive feature of human cognition, but current accounts are incomplete.

We desire a computational theory of problem solving that:

- Retains as many of the core assumptions from the standard framework as possible;
- Accounts for the great variety of strategies observed in humans and machines;
- Explain how domain expertise can reduce search and make problem solving more effective.

In this talk, we present such a revised theory and describe HPS, an implemented architecture that incorporates its tenets.

The Standard Theory

The standard theory of problem solving (Newell & Simon, 1961) makes a number of claims:

- Problem solving involves the mental representation, interpretation, and manipulation of *symbol structures*.
- This process involves *search* through a space of candidates that it encodes as such symbol structures.
- Search is not exhaustive but rather is guided by *heuristics* that make it selective and tractable.
- Problem solvers use *means-ends analysis* to decompose complex problems into simpler ones.
- Expert behavior calls on *knowledge* about a domain to reduce and sometimes even eliminate search.

Evidence from repeated empirical studies has been consistent with most aspects of this theory.

Means-Ends Analysis

As embodied in Newell and Simon's *General Problem Solver*, means-ends analysis relies on four basic ideas:

- Problem solving interleaves *transforming* the current state into a desired one with *applying* an operator to do the transformation.
- Operators are considered only if they *reduce differences* between the current and desired state.
- Problem solving involves search through a space of alternative problem decompositions.
- The selected operator determines the structure of the resulting problem decomposition.

Only the *second* assumption is limiting and problematic. The other three tenets may well be worth retaining.

A Revised Theory of Problem Solving

We propose a revised theory of problem solving that replaces means-ends analysis with three postulates:

- Problem solving involves recursively dividing problems into subproblems, with solutions stated as *decomposition trees*.
- Search details – operator generation / evaluation, node selection, success / failure criteria – are controlled by *strategic parameters*.
- Domain expertise is often encoded as *generalized decompositions* for breaking a problem into subproblems (as in HTNs).

This revision retains the key ideas of means-ends analysis without committing to chaining off goals.

We have embedded these assumptions in HPS, an architecture for *h*ierarchical *p*roblem *s*olving.

Encoding Problem Decompositions

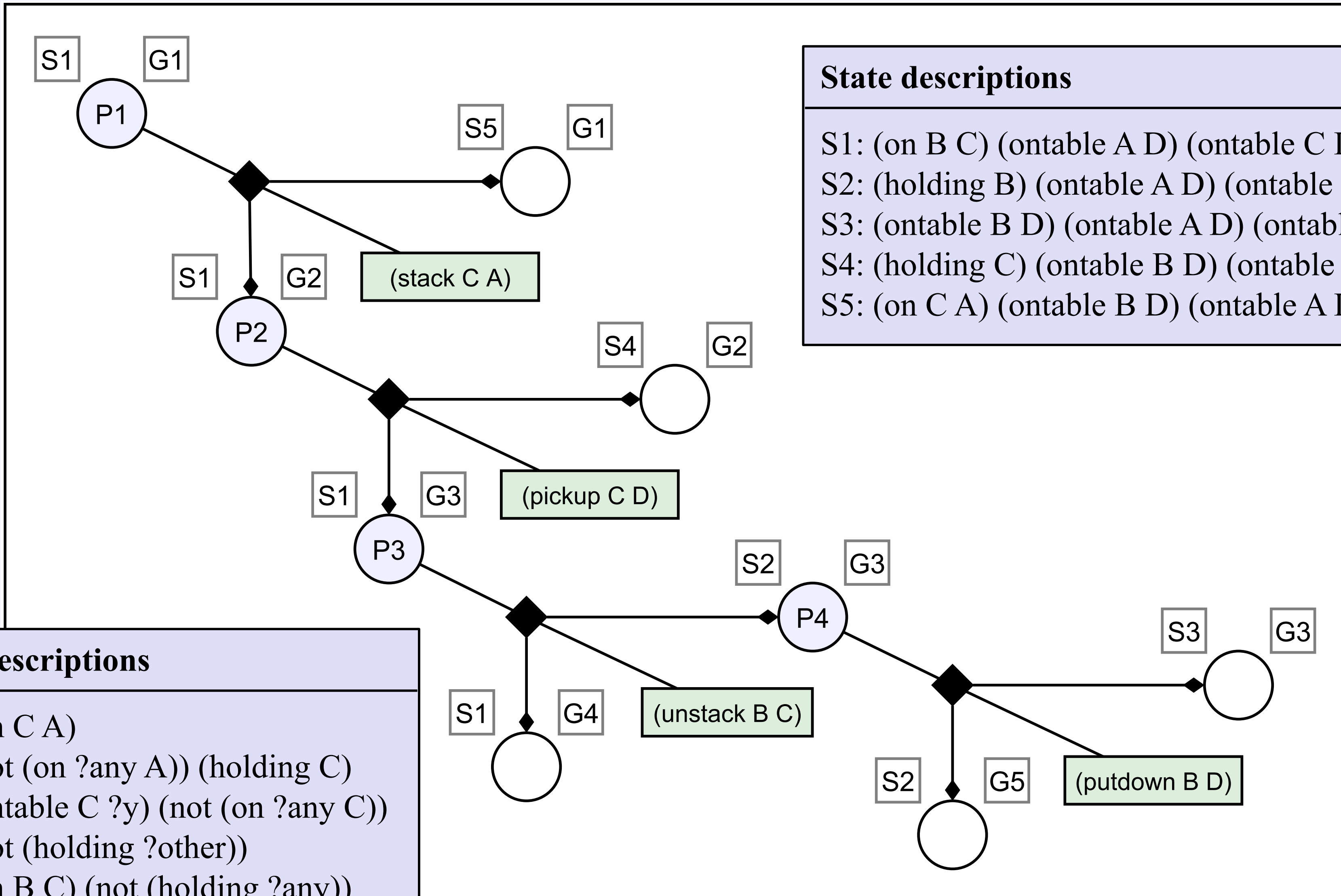
The HPS architecture encodes problem solutions – both partial and complete – as decomposition trees.

Each element in such an AND tree has two components:

- A *problem*, which includes a *state* and a *goal* description;
- A *decomposition*, which specifies an operator instance that breaks the problem into:
  - A *down* subproblem, which must be solved before applying the operator instance;
  - A *right* subproblem, which must be solved afterward to achieve the parent's goals.

Each operator instance has application conditions, expected results, and constraints on shared variables.

Encoding Problem Decompositions



State descriptions

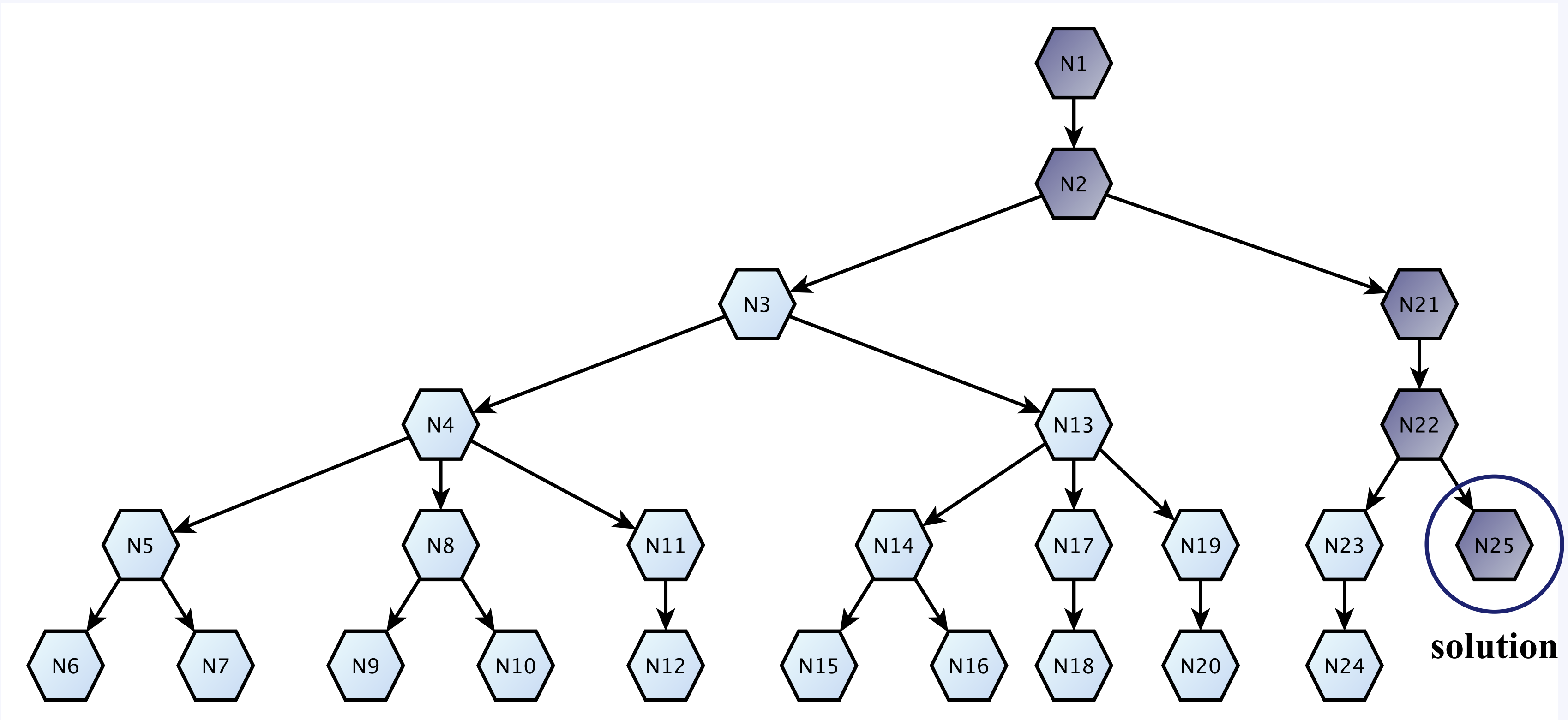
S1: (on B C) (ontable A D) (ontable C D)
S2: (holding B) (ontable A D) (ontable C D)
S3: (ontable B D) (ontable A D) (ontable C D)
S4: (holding C) (ontable B D) (ontable A D)
S5: (on C A) (ontable B D) (ontable A D)

Goal descriptions

G1: (on C A)
G2: (not (on ?any A)) (holding C)
G3: (ontable C ?y) (not (on ?any C))
(not (holding ?other))
G4: (on B C) (not (holding ?any))
G5: (holding B)

Organizing Nodes into Search Trees

Here is a search tree HPS generates during its problem solving, with nodes along a successful path shaded.



Each node (partial plan) elaborates its parent by adding a new subproblem decomposition. Numbers reflect generation order.

The HPS Problem-Solving Cycle

HPS operates in cycles that take one of five meta-level actions:

- *Option 1*. If there are enough acceptable plans or no further choices, then halt and return these solutions.
- *Option 2*. If the plan for the current node is acceptable, then store it and select another node in the search tree.
- *Option 3*. If the current plan is unacceptable (e.g., involves a loop), then mark it as failed and select another node in the search tree.
- *Option 4*. If the current plan has no operator for focus subproblem *F*, then generate operators for F and calculate their evaluation scores.
- *Option 5*. If the current plan has untried operators but subproblem *F* has none, then select one and elaborate the plan by decomposing *F*.

The *focus* problem is an 'open' task in a hierarchical plan that HPS does not yet consider solved.

Modulating the Search Process

HPS incorporates ten parameters that determine its choices during problem solving:

- *Option 1*. When checking whether to halt, HPS calls on a parameter that sees if it has found enough acceptable plans.
- *Option 2*. A parameter tests for plan acceptability (e.g., if all goals satisfied); if so, another picks the next node to visit (e.g., the parent).
- *Option 3*. A fifth parameter tests the plan for failure (e.g., if a loop has occurred); if so, another one chooses the next node to visit.
- *Option 4*. A parameter generates operator instances (e.g., forward or backward chaining); another computes a score for each candidate.
- *Option 5*. Final parameters select an operator to decompose the focus problem, score the resulting elaboration, and select the next node.

These settings are intrinsically *composable*, so that different combinations can reproduce many distinct strategies.

Encoding and Using Domain Expertise

Strategic parameters support different varieties of problem solving, but not expert behavior; to this end, HPS can:

- Store domain-specific knowledge on how to decompose classes of problems into subproblems, with each ‘method’ stating:
  - An associated task name, conditions, and optional effects;
  - A *down*, *main*, and *right* subtask that must be carried out.
- Use this knowledge during ‘operator’ generation and evaluation;
  - On entering a hierarchical method, considers only candidates with relevant task names and matched conditions;
  - On completing a method, returns to knowledge-lean search.

HPS can solve problems stated only as tasks, only as state-goal pairs, or as their combination, much like UPS (To et al., 2015).

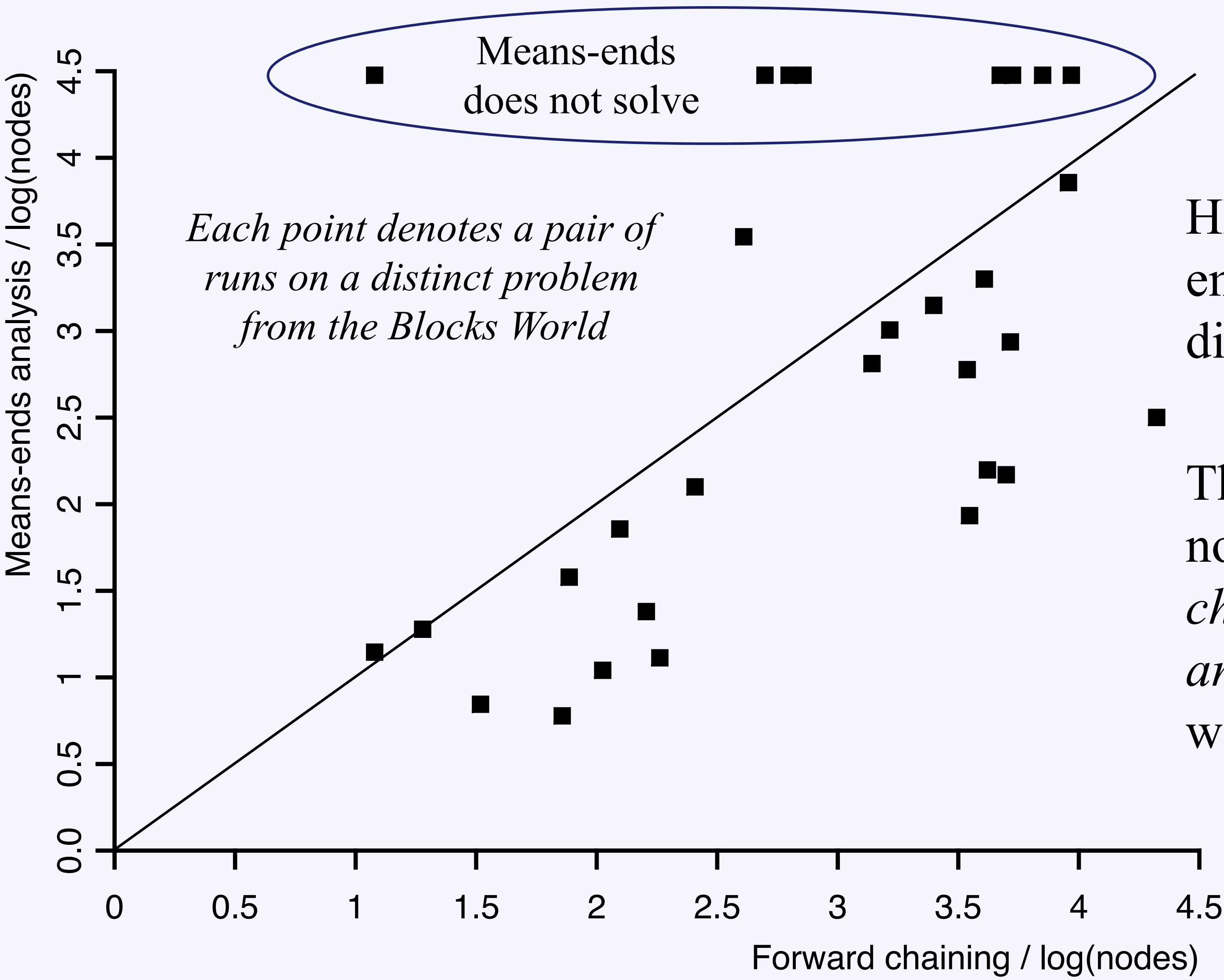
Basic Problem-Solving Ability

We have demonstrated HPS's ability to solve novel problems on two familiar domains:

- *Blocks World* – finding plans that produce desired configurations:
  - Solutions to problems ranged from four to 12 steps.
  - Depth-first forward chaining solved 30 out of 30 tasks.
  - Depth-first means ends solved only 23 of these problems.
- *Kinship inference* – deducing complex relations from simple ones:
  - Problems required from one to eight inference steps.
  - Forward chaining could not solve more complex tasks.
  - Goal-driven means-ends analysis had little difficulty.

These basic results show that strategy effectiveness interacts with domain characteristics.

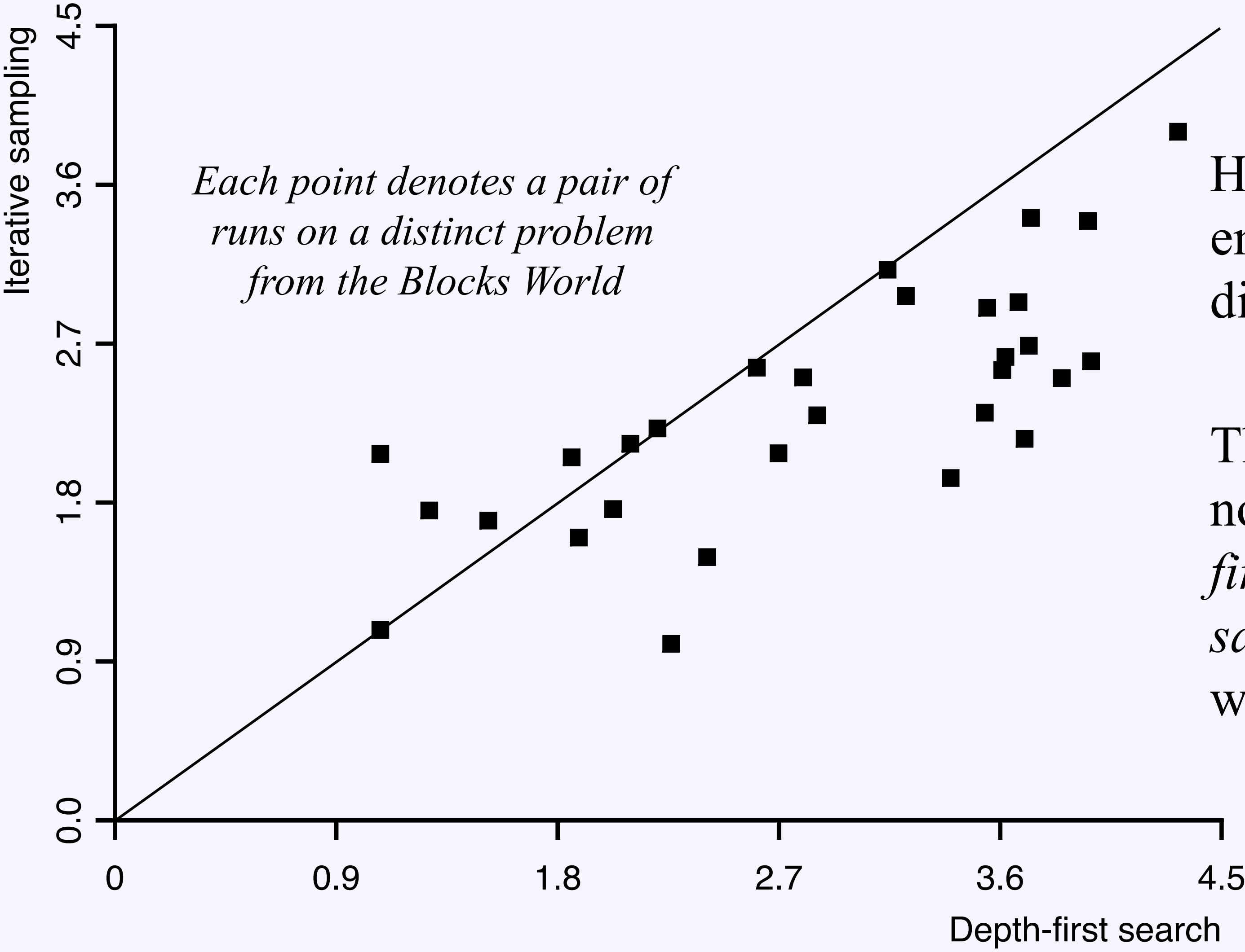
Forward Chaining vs. Means-Ends



HPS parameters support empirical comparison of different strategies.

This study compares the nodes visited by *forward chaining* and *means-ends analysis* when combined with depth-first search.

Depth-First Search vs. Iterative Sampling



HPS parameters support empirical comparison of different strategies.

This study compares the nodes visited by *depth-first search* and *iterative sampling* when combined with forward chaining.

Benefits of Domain Knowledge

Our final study examined the ability to use domain-specific decompositions to reduce search.

We showed that HPS solves Blocks World problems efficiently using hierarchical methods, including recursive ones:

- When stated in terms an initial state and a task to accomplish;
- When specified using an initial state, goals, and a task;
- When stated using an initial state and goals but no task.

The first two conditions required little search; the third took more but still less than without domain knowledge.

The second and third settings exceed the abilities of traditional HTN planning systems.

Relations to Earlier Research

How do our theoretical claims, and their embodiment in HPS, relate to previous research?

- *Problem solutions are represented by decomposition trees*.
  - Widely adopted in both theorem proving and HTN planning.
- *Strategic content underlies variation in problem solving*.
  - Soar uses meta-level control to implement different strategies.
  - HPS is closer to Prodigy, but has substantially greater coverage.
- *Domain expertise takes the form of generalized decompositions*.
  - In HTN planning, but seldom joined with primitive operators.
  - Other forms of knowledge, like rejection rules, are possible.

We have drawn on these earlier ideas but also *combined* them to produce an extended theory of problem solving.

But What About *Soar*?

Soar (Laird, 2012) can reproduce the same strategies as HPS, but they offer different *theories* of problem solving:

- HPS makes stronger assumptions about structures and processes;
- Just as Soar makes stronger assumptions than EPIC or even Lisp.

The ability to generate the same behaviors is *not* equivalent to providing the same theoretical account.

- Soar does not offer *architecture-level* support for storing multiple problem decompositions;
- One could write Soar programs that do so, but this extra layer of interpretation would slow processing.

The tradeoff is that HPS must store a search tree of different decompositions, which may have other processing costs.

Plans for Future Work

Our revised theory, and the HPS system, offer an improved account of problem solving, but future work should:

- Extend the framework to reproduce regression planning and abstraction.
- Support more *adaptive problem solving* by:
  - Collecting meta-level data (e.g., relative branching factors)
  - Making parameter settings conditional on these statistics
- *Learn decompositions* from successful problem solving by:
  - Recording traces of solutions for particular problems;
  - Storing generalized versions of these traces for future use.

The result will be an even more flexible and inclusive account of problem solving in cognitive systems.

End of Presentation