Advances in Cognitive Systems 4 (2016) Submitted 4/2016; published 6/2016

# Variations on a Theory of Problem Solving

**Pat Langley** PATRICK.W.LANGLEY@GMAIL.COM
**Chris Pearce** CPEA144@AUCKLANDUNI.AC.NZ
**Yu Bai** YBAI181@AUCKLANDUNI.AC.NZ
**Charlotte Worsfold** CWOR015@AUCKLANDUNI.AC.NZ
**Mike Barley** MBAR098@CS.AUCKLAND.AC.NZ
Department of Computer Science, University of Auckland, Private Bag 92019, Auckland 1142 NZ

**Abstract**

In this paper, we review the standard theory of human problem solving developed by Newell, Shaw, and Simon, along with limitations that have emerged since its introduction. We argue that the theory's emphasis on means-ends analysis is problematic, in that people may use a variety of other strategies to solve novel tasks. However, some aspects of their account remain valid and we present a revised theory that retains and builds on these features. We also describe HPS, a problem-solving architecture that implements the theory by combining search through a space of hierarchical problem decompositions with strategic parameters to produce different behaviors. We demonstrate that the architecture can utilize alternative techniques to solve novel problems and also that it uses domain knowledge, when available, to make search tractable. We conclude by discussing related frameworks for problem solving and our plans for future research in this area.

## 1. Introduction

The ability to solve novel problems is one of the distinctive features of human cognition, and studies of problem solving played a key role in AI's inception. Early models of this phenomenon offered some of the first evidence for the computational character of thinking, and they remain some of the most compelling accounts in the literature on mental processing. However, in recent years there has been little research on models of problem solving that make contact with the psychological literature, and the AI planning community has divorced itself from such matters. Yet current theories of human problem solving, although largely accurate, remain incomplete and we need more comprehensive models that provide deeper insights into this central facet of cognition.

In this paper, we report progress on this scientific front. We review the standard theory of problem solving, developed between the late 1950s and late 1970s, along with its limitations. We focus on means-ends analysis, which humans use in some settings but not others, and on the great variability in problem-solving strategies. We then propose a revised theory that retains some important features of means-ends processing, after which we describe HPS, an architecture for *h*ierarchical *p*roblem *s*olving that embodies the new theory's postulates. Next we report on HPS's behavior on two task domains, focusing on its support for multiple strategies and its ability to combine knowledge and search. In closing, we discuss related research, reiterate our framework's novel contributions, and outline directions for future work.

## 2. Theories of Human Problem Solving

Langley and Rogers (2005) have reviewed what they call the *standard theory of problem solving*, proposed initially by Newell, Shaw, and Simon (1958) and extended over the next 20 years. This framework explains, in computational terms, how humans solve unfamiliar problems. Early work was limited to abstract puzzles like the Tower of Hanoi and proving logic theorems, while later research also examined expert behavior on tasks like solving physics problems.

The standard theory of problem solving makes a number of claims, increasingly specific, about the character of human cognition:

- Problem solving involves the mental representation, interpretation, and manipulation of *symbol structures*, an idea Newell and Simon (1976) refined into their *physical symbol system* hypothesis.
- The problem-solving process carries out *search* through a space of candidates that it encodes as symbol structures and that it generates and tests by symbol processing.
- This search is not exhaustive but rather is guided by *heuristics* that make it selective; combined with the second claim, Newell and Simon referred to this as the *heuristic search* hypothesis.
- Problem solvers often utilize *means-ends analysis*, which uses differences between current and desired states to select operators and create subtasks that organize search.
- Expert problem solving utilizes knowledge about a domain to reduce and sometimes even eliminate search, shifting behavior from backward chaining to forward chaining (Larkin et al., 1980).

Evidence from repeated empirical studies has been consistent with many aspects of this account of problem solving. There is wide agreement that high-level cognition relies on symbol processing and heuristic search through a problem space, and that expertise can reduce search substantially.

The situation differs for means-ends analysis, which has two key limitations. One problem is that, as typically defined,[1] this method cannot solve certain tasks that involve interactions among goals (Sussman, 1975). Means-ends analysis has difficulty with these 'anomalies' because it only considers operators that achieve one or more goals. Despite its many advances, the AI planning community abandoned it during the 1980s to focus on other schemes, at first partial-order methods (Kambhampati, 1997) and more recently forward-chaining techniques (Hoffmann & Nebel, 2001). The second drawback, not widely documented in the literature, is that humans utilize means-ends analysis in some contexts but not others. For example, the strategy is observed in novice behavior on the Tower of Hanoi puzzle, geometry theorem proving, and physics problems. However, in games like chess and checkers, people carry out forward search from the current state, using techniques like progressive deepening (de Groot, 1978), which involves repeated greedy exploration of the space.

Human problem solvers also vary along other dimensions. They often halt when they find a single solution, but in some cases they obtain multiple answers and select the best. On simple games like Tic-Tac-Toe that involve small spaces, they may carry out exhaustive depth-first search, whereas on complex games like chess, they use methods like progressive deepening that make fewer memory demands. On traditional puzzles, they accept only solutions that achieve all goals, but on real-world tasks that involve tradeoffs, they often accept ones that reach only a subset of goals. Again, such differences are seldom discussed in the literature, but they are commonly acknowledged.

1. Laird et al. (1987) adopt a somewhat different definition for means-ends analysis than the one we use here.

These findings suggest that means-ends analysis, at least as presented in the General Problem Solver (Newell, Shaw, & Simon, 1960), should not remain a central element in the theory. But the GPS account has two important features that, by themselves, appear worth retaining. First, it relies on the recursive decomposition of problems into subproblems, with search focusing on which decomposition to select. Second, it utilizes two ideas — *transforming* one state into another that satisfies a set of goals and *applying* an operator to reduce differences between the current and desired state — that reflect distinct facets of problem solving. We should not throw out these promising infants with the dirty bath water.

We propose a revised theory of problem solving that replaces the claim about means-ends analysis with two new postulates, along with a third that complements them:

- Problem solving involves the recursive decomposition of problems into subproblems, with solutions taking the form of decomposition trees.
- Details of search behavior, including operator generation and evaluation, node selection, and criteria for success and failure, are determined by strategic parameters.
- Domain expertise often takes the form of generalized decompositions that specify how to break a problem into subproblems and that play the role of higher-level operators.

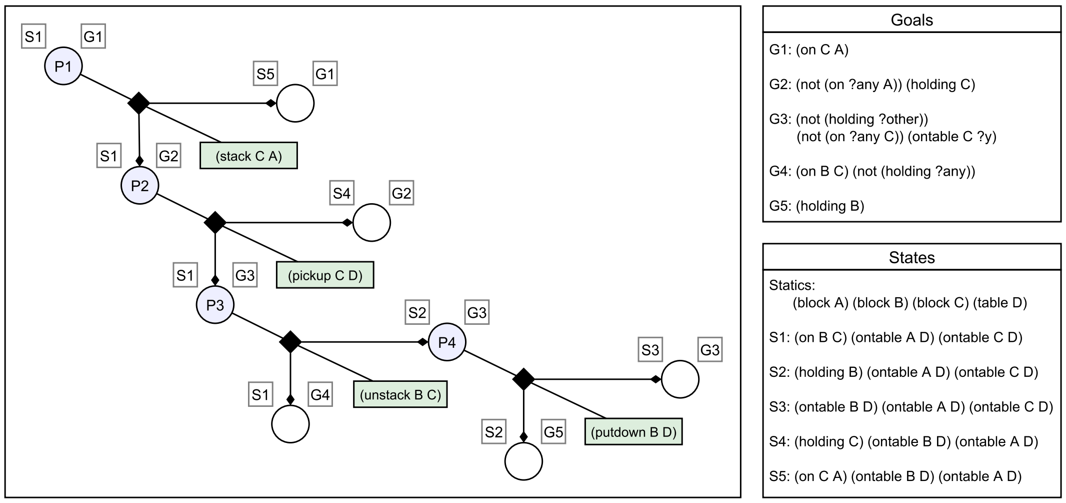
This variation on the standard theory retains the key ideas of means-ends analysis without its commitment to chaining only off operators that achieve goals. It replaces this control scheme with strategies that specify how to make such choices, with traditional means-ends analysis being a special case. Thus, the new theory is less constrained than its predecessor but is consistent with the variability observed in human problem solving. On the other hand, it includes a more specific claim about the structure of domain knowledge, stating that this takes the form of generalized decompositions. This make a direct connection with work in the AI planning literature on hierarchical task networks (e.g., Nau et al., 2003) and with the psychological literature on the hierarchical organization of skills (e.g., Miller, Galanter, & Pribram, 1960). However, the theory as stated is abstract and it would benefit from a computational incarnation that makes its postulates operational.

## 3. The HPS Architecture

We have incorporated these theoretical ideas into HPS, an implemented architecture for *h*ierarchical *p*roblem-*s*olving. In this section we describe the system in terms of its representation of problem solutions and search trees, its basic cognitive cycle, and its reliance on strategic and domain knowledge to guide processing. We will argue later that HPS makes stronger theoretical commitments than other architectures like Soar (Laird et al., 1987), which require additional knowledge and constraints to reproduce our system's range of behaviors. Soar itself makes stronger claims than an architecture like EPIC (Kieras & Meyer, 1997), but we hold that HPS imposes even stronger constraints, which is a desirable feature of a scientific theory.

### 3.1 Representing Problem Decompositions and Search Trees

Recall that one of our key postulates is that problem solutions — both partial and complete — are encoded as problem decomposition trees. Each AND element in such a tree has two components. The *problem*, which includes a *state description* and a *goal description*, plays the same role as



Goals

G1: (on C A)

G2: (not (on ?any A)) (holding C)

G3: (not (holding ?other))
(not (on ?any C)) (ontable C ?y)

G4: (on B C) (not (holding ?any))

G5: (holding B)

States

Statics:
(block A) (block B) (block C) (table D)

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)

*Figure 1.* One possible solution for the problem P1 that involves four decompositions (diamonds) into subproblems (circles), each with an associated operator (rectangles). Every subproblem has an associated state (to its left) and goal description (to its right). Empty nodes denote degenerate problems whose goals match their state. This plan includes four steps: unstack B from C, putdown B, pickup C, and stack C on A.

a ‘transform’ element in the General Problem Solver but uses a STRIPS-like notation. Both descriptions can include literals about the world, such as *(on A B)*. State elements denote the current situation, while goal literals describe a class of desired states. Figure 1 presents an example from the Blocks World in which states are depicted graphically and goal descriptions are stated as literals. The figure also shows nine linked problems, each of which refers to a named state on the left and a labeled goal description on the right.

The other component of a problem solution is its *decomposition*, denoted by black triangles, which play the same role as ‘apply-operator’ elements in the GPS system. Each decomposition specifies both an instantiated domain operator and how to break its problem into subproblems. These include a *down* subproblem, which must be solved before applying the operator, and a *right* subproblem, which must be solved afterward to achieve the parent’s goals. Figure 1 shows four operators, with two down subproblems (*P2* and *P3*) that appear below their parents and one right subproblem (*P4*) that appears on its parent’s right. Each operator instance, such as *(stack C A)*, has application conditions, expected results, and constraints on variables shared by these structures. The terminal nodes have no associated decompositions because their states match their goal descriptions.

HPS decompositions are similar in structure to those used by GPS, but we will see that it generates them in a different manner. As a result, our approach subsumes other frameworks that, at first glance, do not appear to involve problem decomposition. For instance, an entirely right-branching decomposition tree maps onto a solution found by forward search, whereas an entirely down-branching tree maps onto the results of regression planning (McDermott, 1991). In contrast, means-ends produces mixed structures like the one shown in the figure, with some subproblems of each type. Different types of decompositions emerge from different problem-solving strategies.

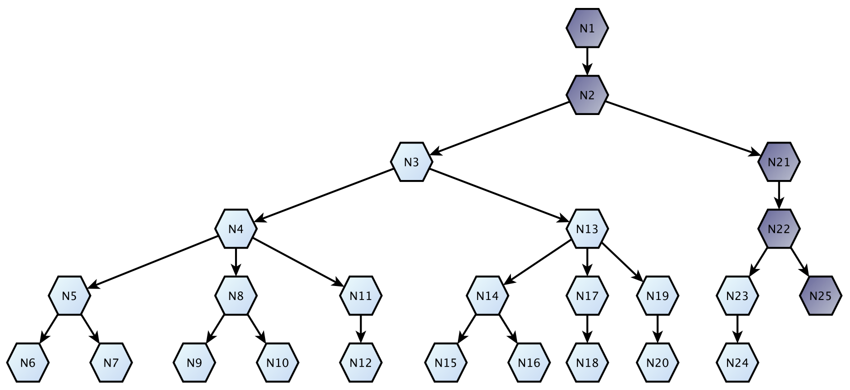


*Figure 2.* A search tree that HPS generates during problem solving, with nodes along a successful path shaded and with *N25* denoting the decomposition in Figure 1. Each node elaborates on its parent (immediately above it) by adding a new subproblem decomposition. Numbers reflect the order in which nodes were generated.

The construction of such hierarchical plans requires HPS to select among alternative decompositions, and it must keep these choices distinct in some manner. To this end, the system organizes candidate solutions in an OR search tree, as shown in Figure 2. Each node in this tree denotes a decomposition tree like that in Figure 1, although most are partial breakdowns that do not provide complete solutions. For example, the root node *N1* contains only the top-level problem *P1* from Figure 1, *N3* specifies the decomposition of *P1* into *P2* and the operator *(stack C A)*, and *N21* contains the decomposition of *P2* into *P3* and the operator *(pickup C D)*. Each node also introduces a new state that satisfies one of its subproblem's goals. The final node along the shaded path, *N25*, corresponds to the complete decomposition in Figure 1, which encodes a plan with four steps. A node in this search tree may have zero or more children, where each child elaborates on its parent by introducing a new problem decomposition.[2] If a node has multiple children, then each contains a different elaboration on its parent's partial solution.

## 3.2 The HPS Problem-Solving Cycle

The structures just described are central to HPS's problem-solving process. At the outset, the root node contains a single problem with state and goal descriptions. The architecture recursively decomposes this problem into subproblems, adding new nodes to its search tree on the way. This continues until it finds enough solutions to satisfy its success criteria or abandons the effort. Problem solving operates in discrete cycles, with each taking a conditional meta-level action that HPS considers in a specific order:

1. If the system has found enough acceptable plans or if no further choices are available, then it halts and returns these solutions.
2. If HPS decides the plan associated with the current node (plan) is acceptable, then it adds the plan to its solution set and selects another node in the search tree for attention.

2. The content stored with a given node changes over the course of problem solving, as HPS applies operators and uses their results to populate subproblems, but such steps do not produce new nodes in the search tree.

3. If the system finds that the current node (plan) is unacceptable, then it marks it as failed and selects another node in the search tree for attention.
4. If the current node (plan) has no associated operator instances relevant to the focus subproblem *F*, then HPS generates candidates relevant to *F* and calculates evaluation scores for each one.
5. If the current node (plan) has untried operator instances and there is no method yet associated with its focus subproblem *F*, then the system selects an untried alternative and elaborates the current node by decomposing *F*.

The notion of a *focus* subproblem is central to the final two rules. This refers to a terminal subproblem in a node's hierarchical plan that HPS does not yet consider solved. This may involve a down subproblem or a right subproblem, with the latter becoming the focus only after the former has been solved. The architecture extends a given node *N* by decomposing the subproblem *F* that is currently in focus; this step automatically leads to a child of *N* that elaborates its parent.

In summary, problem solving in HPS cycles through five meta-level rules that check for enough acceptable plans, decide if the current node is acceptable or unacceptable, generates operator instances, and selects a generated instance, which extends the search tree. We have described this process as nondeterministic, but the means-ends behavior found in GPS is an important instance of its operation. Now let us consider how HPS reproduces not only that problem-solving strategy but other search techniques as well.

## 3.3 Modulating Problem-Solving Behavior

The structures and processes described in the previous sections provide a foundation for problem solving, but they do not specify which decisions to make. To produce specific behavior, HPS requires some way to select among alternatives at each choice point. The architecture draws on two forms of content: *strategic parameters* that support generic problem-solving strategies and *domain knowledge* that encodes particular decompositions. We discuss each in turn, along with the ways they modulate the architecture's behavior.

When humans encounter novel problems, they employ some strategy to organize their search for a solution. For some tasks, like the Tower of Hanoi, they typically work backwards from the goal; in others, like chess, they typically work forward from the current state. In some contexts people may be satisfied with a single solution, while in others they may continue until they find a number of answers. Such schemes are not concerned with particular domain predicates, and they can be adapted to almost any situation. The flexibility of human problem solvers, and their ability to use such strategies, is one of the phenomena we seek to explain, although we are more concerned with reproducing their general abilities than matching their cognitive behavior in detail.

To account for this ability, HPS incorporates elements that determine behavior at each decision stage. This information takes the form of domain-independent parameters, each associated with one step in the problem-solving cycle and referring to functions that map features of the current node onto one type of decision. These functions are domain independent in that they inspect aspects of the problem's state, goals, operators, and subproblems, but they do not examine domain predicates. For example, a function might mark a node as unacceptable if its depth in the search tree exceeds a particular limit or if its state repeats that in an ancestor. Another might redirect the problem solver to a node's parent upon deciding it is unacceptable.

Strategic elements influence every stage of HPS's processing cycle. The current version of the system incorporates ten such parameters that are best considered in terms of the stage at which the decisions they control come into play:

1. When checking whether it has found enough solutions, HPS calls on a function that sees whether the number of acceptable plans has reached a target. This parameter can range between one and any other positive integer.
2. The architecture invokes another parameter to decide whether the current plan is acceptable. The runs in this paper assume that the final state must satisfy all goals associated with the top-level problem, but HPS also supports satisficing options in which a specified percentage of the goals are satisfied or in which the summed value of matched goals exceeds a threshold. This stage also calls on a parameter that selects another node to examine after finding the current plan acceptable. This defaults to the current node's parent, but another option is to select the root node.
3. When deciding whether the current plan is unacceptable, the system uses a parameter whose associated function inspect the plan's structure for undesirable features. The current version includes options for rejecting nodes that exceed a specified depth in the search tree, that repeat states or goals in ancestor nodes, or their combination. This decision stage also includes a parameter that determines which node to select after failure. The default is the current node's parent, but a different setting instead selects the root note.
4. During the fourth decision stage, HPS utilizes a sixth parameter whose function generates operator instances that are relevant to the current focus subproblem *F*. Implemented options include operators whose conditions match *F*'s initial state, whose results match one or more of *F*'s goals, or the conjunction of these two tests. Another parameter calculates evaluation scores for each generated candidate. Besides the default of generation order, the options here include favoring operator instances that achieve more goals, fewer unmatched conditions, or their combination.
5. The architecture uses an eighth parameter when selecting which operator instance to use in elaborating the current node through further decomposition. The implemented default is to select the candidate with the highest score, but one can imagine other schemes, such as probabilistic sampling. Two final parameters also come into play during the stage. One is responsible for assigning an evaluation score to the elaborated plan. Here the default is to give all nodes a constant value, but a second option is plan length. The other parameter handles node selection after elaboration. The default function chooses the new node itself, but another setting is to select the parent.

In summary, strategic parameters influence HPS's behavior at each stage of its processing. Their settings are inherently composable, so that different combinations of functions can reproduce a wide range of search strategies. This underlies the architecture's ability to account for the great variation observed in human problem solving, and we view the ten parameters to be important elements of the theory that the HPS architecture embodies.

### 3.4 Representing and Using Domain Expertise

Strategic parameters provide HPS with the ability to carry out different forms of novice-level problem solving, but not to reproduce expert behavior, which often shows little apparent search. For this purpose, the architecture also supports the storage and use of domain-specific knowledge about how to decompose certain classes of problems into subproblems. These take a form similar to the ‘methods’ in hierarchical task networks (HTNs), with each stating how to break a named task into ordered subtasks, along with the generalized conditions under which it is appropriate. They differ from methods in traditional HTN planners like SHOP2 (Nau et al., 2003) in that they may also state effects, although some versions of Icarus (Li et al., 2012) also incorporate this facility.

Another representational difference is that hierarchical methods in HPS include at most three subtasks. These correspond directly to three elements that appear in basic decompositions — down subproblems, operator instances, and right subproblems — but they instead specify tasks rather than goals, as well as the order in which to address them. A hierarchical method must include a ‘main’ subtask, but either the down or right subtasks may be empty. However, such a method must always specify at least two subtasks or it would not produce a decomposition.

HPS draws upon this domain expertise in the fourth stage of decision making, when it generates operator instances, and the final stage, when it elaborates the current plan. When hierarchical methods are available, the architecture gives them the same consideration as primitive operators. The system can retrieve only those method instances with matched conditions or with results that achieve goals. However, instead of applying a selected operator when its conditions are met, HPS uses the method to generate subtasks that shift it into a different mode of operation similar to HTN planning. This still involves search, but it is constrained to consider only methods and operators that are indexed by the relevant task. The current implementation is limited to forward chaining once it enters this mode, as in most HTN planners, but it returns to normal operation on successful method completion. Whether HPS uses a stored method or an operator to decompose a problem, the result is an elaborated node that provides more details about the candidate solution. This is consistent with our claim that problem solving utilizes hierarchical decomposition.

Moreover, the architecture can use these hierarchical methods in three distinct ways. If the top-level problem is specified in terms of a state and goals, then it treats methods as if they were primitive operators, using the same strategic control scheme to select among them. As with macro-operators, they let HPS take larger steps through the problem space at the cost of increased branching factors, which can be offset with heuristic guidance, such as preferring candidates that achieve more goals. If the top-level problem specifies a named task, such as clearing a block, then it mimics traditional, task-driven HTN planning. Alternatively, if HPS encounters a problem that stipulates both tasks and goals, it only considers methods that are relevant to both the task *and* the goals. Thus, it combines the abilities of both goal-oriented and task-oriented planners, in much the same way as To, Langley, and Choi’s (2015) UPS system, but with considerably more strategic flexibility.

## 4. Behavior of the HPS System

We have examined the architecture’s behavior in a number of domains to ensure that it operates as intended. We have emphasized planning tasks here, but the system should support other varieties of problem solving, such as design, scheduling, and theorem proving. Our aim was to demonstrate that HPS has the ability to solve novel problems, that it supports a wide range of strategies, and that it utilizes domain expertise, when available, to make problem solving tractable. We have not tried to replicate results from individual studies with humans or compare the system’s behavior to that reported in the psychological literature.

### 4.1 Basic Problem-Solving Ability

To test HPS's ability to solve novel problems, we encoded primitive operators for two domains — Blocks World planning and inferring kinship relations. The first is a well-studied stacking puzzle in which one must create target configurations using operators for picking up a block from the table, putting one down, stacking one block on another block, and unstacking a block from another. The operators for this domain alter the environment, leading relations such as *(on A B)* to become true or false as the plan proceeds, which in turn can lead to interactions among goals. The second domain is quite different, in that one must draw inferences about long-distance relations, such as *(uncle Bob Dan)* and *(grandparent Abe Dan)*, from primitive relations like *(parent Abe Bob)* and *(male Bob)*. Here each operator adds only one literal, with elements never being deleted.

We provided HPS with the operators for each domain, along with settings for strategic parameters that caused arbitrary selection of operators and termination upon finding a single solution. We also set the architecture's parameters to carry out forward chaining combined with depth-first search to a specified level. We ran HPS on ten Blocks World problems with solutions that varied from four to 12 steps, as well as ten kinship reasoning tasks that required proof trees from one to eight inference steps. Given these settings, the system found solutions, sometime nonoptimal, for all but one of the Blocks World tasks, which it had not completed after generating more than 2,500 nodes. The architecture solved the simpler inference tasks without difficulty, but it could not find some of the more complex proofs in the time allowed. Despite the existence of many paths that produce a given inference, the combination of unguided forward chaining, depth-first search, and high branching factor made these problems intractable. In contrast, means-ends analysis' reliance on goal-driven operator selection let it handle all ten problems within the time available.

### 4.2 Variation in Problem-Solving Strategies

We have carried out exploratory studies with HPS's strategic parameters to confirm that it can reproduce a number of familiar search techniques, as well as ones that do not appear in the literature. For example, the architecture can return to a parent node when one of its children is deemed unacceptable, but shift to the root node upon finding a solution. This strategy, which combines aspects of depth-first search and iterative sampling (Langley, 1992), should produce more diverse solutions than the former alone. However, the framework has enough parameters that a systematic study of their interactions will be time consuming, so here we instead examine only a few variations.

Modifying the parameter settings for operator selection lets HPS reproduce simple forward-chaining techniques and traditional means-ends problem solving. By holding the other decision functions constant, we can carry out meaningful experiments that compare the different methods' behaviors within the same architecture. Figure 3 presents a scatter plot that compares the number of nodes generated by each technique on ten different problems from the Blocks World when combined with uninformed depth-first search. This reveals that neither approach dominates on all problems, although means ends finds solutions with less effort than forward chaining in a majority of this small sample. Analyses of these problems' characteristics, such as their branching factor in the forward and backward directions, may clarify when one method will be more effective than the other.

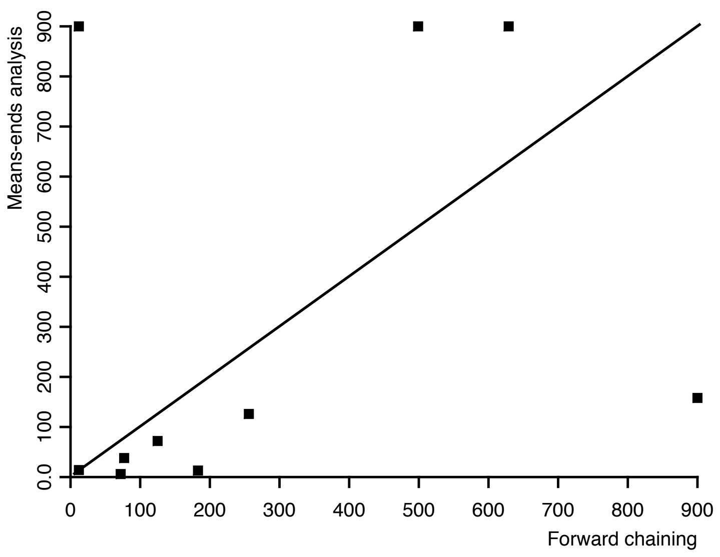


*Figure 3.* A scatter plot that compares the number of nodes generated by HPS on ten Blocks World problems when combining depth-first search with forward chaining and with means-ends analysis. Points below the diagonal denote problems in which means-ends analysis visited fewer nodes. Items at the *x* and *y* boundaries indicate problems that one of the methods failed to solve in the time allocated.

We also used HPS to compare the relative effectiveness of depth-first exploration and the lesser known iterative sampling, which carries out repeated greedy search from the root node.[3] Figure 4 presents another scatter plot that displays the number of nodes generated by each scheme, when combined with forward chaining, on the same ten Blocks World problems. Some points fall above the diagonal, but in many cases iterative sampling examines fewer partial plans that depth-first search, despite the latter's greater familiarity. However, the sample is small, and in any case we are less interested in which technique fields a majority than in understanding the conditions that underlie their effectiveness. Such insights will be necessary before we can augment HPS to adapt its strategies to the problems it encounters, which is one of our long-term aims.

## 4.3 Problem Solving with Domain Expertise

Finally, we wanted to show that HPS can use domain knowledge, stated as generalized decompositions, to solve problems more effectively than without such content. For this purpose, we created domain-specific methods for five tasks in the Blocks World. These included simple hierarchical methods for picking up and stacking a block, for unstacking a block and putting it down, and for unstacking and stacking a block. We also provided HPS with more sophisticated decompositions, similar to those in hierarchical task networks, for clearing a block in a tower and for inverting all

3. Langley's (1992) initial analysis of iterative sampling assumed that later passes retained no memory of previous ones. Because HPS stores the parent-child relations of nodes it has examined, our version revisits nodes that were created on earlier rounds without duplicating them.

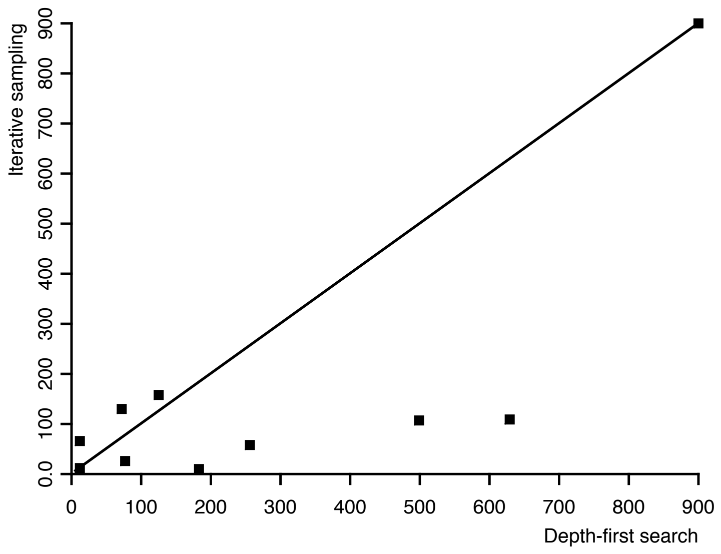


*Figure 4.* A scatter plot that compares the number of nodes generated by HPS on ten Blocks World problems using forward chaining with depth-first search and with iterative sampling. Points below the diagonal denote problems in which iterative sampling visited fewer nodes.

blocks in a tower. Knowledge about the latter two activities essentially specified recursive procedures for carrying out these tasks in terms of recursive and base cases.

Given this content, we presented the architecture with five Blocks World problems in terms of an initial state and a task to carry out, as typically done with hierarchical task networks. With strategic parameters set to forward chaining and depth-first search, the architecture solved every problem easily, which was unsurprising because such top-down planning requires very little search when appropriate decompositions are available. We also presented HPS with five problems in the domain that mixed state-like goals and task names. Here only methods that both addressed the specified task *and* achieved the goals could contribute toward a solution. As before, the system solved all five problems with little effort, as the hierarchical knowledge greatly reduced the branching factor during the search process.

A third study examined HPS's ability to benefit from hierarchical domain expertise when given problems with only state-like goals. Here we selected strategic settings that produced forward-chaining, depth-first search and that preferred operators and methods which achieved more goals. We provided HPS with the same hierarchical methods as in the second study, which it again used to solve all problems efficiently. This mode of operation took more search than when we told the system which tasks to solve, but it still found solutions far more rapidly than when it did not have access to the knowledge. The architecture solved a few problems by expanding a single, top-level method, but others required combining hierarchical methods with primitive operators. None of the solutions involved chaining more than a few methods, but this ability nevertheless exceeds that of traditional HTN planning mechanisms.

## 5. Relations to Previous Research

Earlier in the paper, we presented three postulates that, combined with elements of the standard account, make up our revised theory of problem solving. In this section, we discuss the substantial body of previous work that is relevant to each of these ideas, in each case outlining what the earlier efforts share with our framework and how they differ. We also consider how HPS relates to other problem-solving architectures.

The first theoretical tenet is that *problem solutions are represented by decomposition trees*. This is certainly not new; Newell et al. (1958) introduced it in their work on the Logic Theorist. The idea is widely adopted in research on both theorem proving and HTN planning, and computer models of expert human performance in geometry and physics incorporate a similar assumption. These are distinct from means-ends accounts of problem solving, which use a particular strategy to search a space of decompositions (Newell et al., 1960; Carbonell et al., 1990; Li et al., 2012). Our contribution is to postulate that decomposition — along with use of selected operators and methods to generate subproblems — is central to problem solving, but separate from this particular mechanism.[4] Other well-known techniques, like forward search and regression planning, are not usually viewed in these terms, but they are both subsumed by our framework as special cases.

The second theoretical assumption is that, when combined with the decomposition framework, *strategic parameters for decision making support a wide range of problem-solving mechanisms*. HPS's strategic elements let it use structures like those in GPS without restricting them to means-ends analysis. The idea of meta-level control (e.g., Genesereth et al., 1981) also has a long history in AI, but it has not been widely adopted. The most successful meta-level architecture is Laird's (2012) Soar, whose early work emphasized the combintion of 'weak methods' and domain-independent control rules to reproduce a variety of established search techniques (Laird, 1984). Another problem-solving architecture, Prodigy (Carbonell et al., 1990), can fall back on default search methods when domain expertise is unavailable, but papers have not emphasized this ability and it is limited by its commitment to means-ends analysis.

Soar can certainly reproduce each of HPS's strategies, but in some cases this would only be possible by introducing an additional layer of interpretation. The architecture cannot simultaneously encode and store multiple problem decompositions, although one could write Soar programs that, at considerable effort, create and track such alternatives. This would require additional rule-based processing that seems likely to slow problem solving substantially. The tradeoff is that HPS must store a search tree in which each node describes a different hierarchical decomposition, which requires considerable memory and might introduce different processing costs. Experimental comparisons of scalability would clarify this tradeoff further. These observations relate to our earlier claim that HPS provides a stronger, more constrained theory of problem solving than Soar. The ability to reproduce behaviors is not the same as providing a theoretical account. We could mimic all of HPS's strategies in Lisp, but this would not make the latter a comparable theory of problem solving. Soar makes far stronger assumptions than Lisp about structures and processing, but weaker ones than HPS. Whether the latter's tenets are desirable is ultimately an empirical question.

4. Two other systems — PRL (Marsella & Schmidt, 1993) and SteppingStone (Ruby & Kibler, 1993) — also search a space of problem decompositions to solve problems, but rely on quite different forms of knowledge.

Our third theoretical claim is that *domain expertise often takes the form of generalized decompositions for dividing problems into subproblems*. HTN planning systems draw on this idea, but they rely completely on such domain knowledge to solve problems and break down when knowledge is incomplete. This contrasts with our theory, and its implementation in HPS, which utilizes such domain expertise when it is available but combine it with first-principles search when it is only partial or even entirely absent. A few systems, like DUET (Gerevini et al., 2008), GoDeL (Shivashankar et al., 2013), and UPS (To et al., 2015) integrate HTN planning with first-principles search, but they do not support alternative search strategies. The UPS planner comes the closest to HPS in that it treats hierarchical methods as if they are operators until expanding them, when it invokes an HTN-like scheme, but this system was limited to forward search. One could also mimic HPS's approach to problem decomposition in Soar, but the latter also supports other encodings of domain expertise; again, being able to implement an idea is not the same as making it a central theoretical postulate.

Our implementation of the revised theory, HPS, shares some features with earlier frameworks. Other cognitive architectures like ACT-R (Anderson & Lebiere, 1998) and EPIC (Kieras & Meyer, 1997) operate in cycles, but they typically focus on processing production rules, with no special attention given to problem solving. Two notable exceptions are Soar (Laird et al., 1987; Laird, 2012) and Prodigy (Carbonell et al. 1990), both of which emphasize staged processing for problem solving. Soar has phases for elaborating the current problem, selecting an operator to apply, and applying an operator. Prodigy has stages for goal selection, operator selection, and variable binding; it also includes steps for operator generation and backtracking, but it is limited to means-ends analysis and cannot modulate them using strategic knowledge. The FLECS (Veloso & Stone, 1995) extension to Prodigy supports a wider range of search methods, but HPS supports substantially greater flexibility. GIPS (Jones & VanLehn, 1994) and EUREKA (Jones & Langley, 2005) both use flexible versions of means-ends analysis to search a space of decompositions, and they have inspired our work, but neither support the same strategic diversity as our framework.

Each of these earlier systems shares at least some of our theoretical assumptions, but none combines all of them in a single, integrated framework.[5] Prodigy and Soar are the most similar in spirit, with Soar in particular being general enough to implement the same strategies as HPS. However, it does not make explicit theoretical commitments about either the hierarchical structure of solutions and domain expertise or the loci of strategic control that we believe are central to a fuller account of the flexible character of problem solving.

## 6. Concluding Remarks

In this paper, we reviewed the standard theory of human problem solving, including its widely accepted postulates and its more questionable commitment to means-ends analysis. In response, we proposed a revised theory that replaced this assumption with two others: that the process involves recursive decomposition of problems into subproblems and that this activity is controlled by settings of strategic parameters. The former is consistent with observations of human behavior that led to

5. HPS is the successor to an earlier architecture for flexible problem solving, FPS, that had similar aims and incorporated the same basic theoretical assumptions (Langley, Pearce, Barley, & Emery, 2014).

Newell et al.'s General Problem Solver, while the latter explains the variability observed in problem solving that GPS and its descendent Prodigy cannot model.

The new theory also states that domain expertise takes the form of generalized problem decompositions used in the same way as domain operators. This aligns well with the idea that solutions take the form of decomposition trees and with research on hierarchical task networks. However, our framework differs from the latter in that it can combine domain expertise with primitive operators, using knowledge where possible and search where necessary. We also described HPS, a problem-solving architecture that incorporates these assumptions, along with empirical results that demonstrate its ability to solve novel problems, support a variety of stratetegies, and utilize domain knowledge to reduce search. Finally, we examined earlier approaches to problem solving and how they relate to our framework.

We believe that the revised theory, and its implementation in HPS, offers an improved account of problem-solving behavior, but we should also extend them in a number of directions. One limitation is the inability to reproduce either regression planning or abstraction. The first appears to require a new parameter that determines whether goals included in down subproblems come only from operator conditions, as HPS assumes currently, from the parent problem, as in regression planning. Abstraction planning (e.g., Knoblock, 1992) solve problems at higher levels of a decomposition before proceeding to lower ones. We hope to model this ability with a parameter that controls the order of traversal in decomposition trees, supporting breadth-first in addition to depth-first processing. We should also add the ability to consider alternative ways that a state partially satisfies goals, as Langley and Trivedi (2013) have reported in their ICARUS work.

A more important extension would explain how one can adapt strategies to different situations. The architecture supports alternative techniques with different settings for strategic parameters, but a more complete version would switch between strategies as appropriate. This functionality requires access to meta-level information about the state of problem solving, such as branching factors in the forward and backward directions or how many attempts have been made to elaborate a particular node. We must also alter HPS's strategic parameters to make them conditional on such meta-level data. Unlike some techniques for meta-level control (e.g., Genesereth et al., 1981), this approach would collect simple statistics during the course of problem solving and use it reactively. These extensions should let our system adapt its decision making to the current setting, both improving search scalability and explaining more deeply the variations observed in human problem solving.

Future versions of the architecture should also learn useful decompositions from successful problem solving. We hypothesize that hierarchical domain methods are simply generalized traces of successful problem decompositions that are found during search. This suggests that HPS could learn by storing such decompositions in its library of hierarchical methods, using an analytical process to determine their conditions and effects. This approach is similar to the one reported by Li, Stracuzzi, and Langley (2012) in their work on learning in ICARUS. A key issue is to determine which task names to associate with a new method, which they addressed in terms of the goals involved in each problem. Once we have extended the theory and architecture in these directions — especially by introducing adaption and learning — they will provide an even more flexible and inclusive account of problem solving in cognitive systems.

## Acknowledgements

This research was supported in part by Grants N00014-10-1-0487 and N00014-15-1-2517 from the Office of Naval Research, which is not responsible for its contents. We thank John Laird, Manuela Veloso, and Subbarao Kambhampati for providing information about earlier work in the same tradition, as well as Christopher MacLellan and Miranda Emery for their efforts on previous versions of the system.

## References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum.

Carbonell, J. G., Knoblock, C. A., & Minton, S. (1990). Prodigy: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence*. Lawrence Erlbaum.

Gerevini, A., Kuter, U., Nau, D. S., Saetti, A., & Waisbrot, N. (2008). Combining domain-independent planning and HTN planning: The Duet planner. *Proceedings of the Eighteenth European Conference on Artificial Intelligence* (pp. 573–577). Patras, Greece.

de Groot, A. D. (1978). *Thought and choice in chess* (2nd Ed.). The Hague: Mouton Publishers.

Genesereth, M. R., Greiner, R., Smith, D. E. (1981). *MRS manual* (Technical Report HPP-80-24). Department of Computer Science, Stanford University, Stanford, CA.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Jones, R. M. & Langley, P. (2005). A constrained architecture for learning and problem solving. *Computational Intelligence*, *21*, 480–502.

Jones, R. M., & VanLehn, K. (1994). Acquisition of children's addition strategies: A model of impasse-free, knowledge-level learning. *Machine Learning*, *16*, 11–36.

Kambhampati, S. (1997). Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, *18*, 67–97.

Kieras, D., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, *12*, 391–438.

Knoblock, C. A. (1992). An analysis of ABSTRIPS. *Proceedings of the First Conference of AI Planning Systems* (pp. 136–144). Burlington, MA: Morgan Kaufmann.

Laird, J. E. (1984). *Universal subgoaling*. Doctoral dissertation, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Laird, J. E. (2012). *The Soar cognitive architecture*. Cambridge, MA: MIT Press.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1–64.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, *208*, 1335–1342.

Langley, P. (1992). Systematic and nonsystematic search strategies. *Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 145–152). College Park, MD: Morgan Kaufmann.

Langley, P., Pearce, C., Barley, M., & Emery, M. (2014). Bounded rationality in problem solving: Guiding search with domain-independent heuristics. *Mind and Society*, *13*, 83–95.

Langley, P., & Rogers, S. (2005). An extended theory of human problem solving. *Proceedings of the Twenty-seventh Annual Meeting of the Cognitive Science Society* (pp. 1242–1247). Stresa, Italy: Lawrence Erlbaum.

Langley, P., & Trivedi, N. (2013). Elaborations on a theory of human problem solving. *Poster Collection: The Second Annual Conference on Advances in Cognitive Systems* (pp. 111–122). Baltimore, MD.

Li, N., Stracuzzi, D. J., & Langley, P. (2012). Improving acquisition of teleoreactive logic programs through representation extension. *Advances in Cognitive Systems*, *1*, 109–126. Cognitive Systems Foundation.

McDermott, D. (1991). Regression planning. *International Journal of Intelligent Systems*, *6*, 357–416.

Marsella, S. C., & Schmidt, C. F. (1993). A method for biasing the learning of nonterminal reduction rules. In *Machine learning methods for planning*, ed. S. Minton, 499–535. San Francisco, CA: Morgan Kaufmann.

Miller, G. A., Galanter, E., & Pribram, K. A. (1960). *Plans and the structure of behavior*. New York: Holt, Rhinehart, & Winston.

Nau, D., Au, T., Hghami, O., Kuter, U., Murdock, J., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, *20*, 379–404.

Newell, A., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological Review*, *65*, 151–166.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264). UNESCO House, France: UNESCO.

Ruby, D., & Kibler, D. (1993). Learning recurring subplans. In S. Minton (Ed.), *Machine learning methods for planning*. San Francisco, CA: Morgan Kaufmann.

Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). The GoDeL planning system: A more perfect union of domain-independent planning and hierarchical planning. *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (pp. 2380–2386). Beijing.

Sussman, G. J. (1975). *A computer model of skill acquisition*. New York: Elsevier Science.

To, S. T., Langley, P., & Choi, D. (2015). A unified framework for knowledge-lean and knowledge-rich planning. *Proceedings of the Third Annual Conference on Cognitive Systems* (pp. 1–17). Atlanta, GA: Cognitive Systems Foundation.

Veloso, M., & Stone, P. (1995). FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, *3*, 25–52.