Advances in Cognitive Systems (2015) 1-16 Submitted 2/2015; published

Problem Solving through Successive Decomposition

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

Abstract

We present a theory that aims to reproduce the fundamental characteristics of human problem solving, including the ability to use a diverse range of strategies and to incorporate domain expertise when available. To begin, we discuss some known features of problem solving and propose several theoretical assumptions to account for them. Next, we introduce HPS, a hierarchical problem solver that implements these postulates. We describe the system's core structures and processes, and, in each case, explain how they serve our high-level aims. We then describe the different forms of knowledge that can modulate HPS's behavior. To demonstrate that the system accomplishes our objectives, we report three sets of experiments. We conclude by discussing related research and proposing avenues for future study.

1. Introduction

Humans exhibit considerable diversity in the ways that they approach novel problems. They can work forwards from their current state or backwards from their goals; they might consider several alternatives at once or follow one path until they either find a solution or give up. People have many such strategies at their disposal and, since these techniques do not rely on prior knowledge about a particular problem, they can be used to find solutions in almost any domain. However, while these strategies form a key aspect of human problem solving, in the absence of domain expertise to guide or constrain search, they are not very efficient. In contrast, domain experts can solve even complicated problems with ease, and their knowledge drastically reduces the time it takes them to find a solution. Even if their knowledge is incomplete, they can still fall back on their partial knowledge and use search to compensate for the gaps.

As these are intrinsic aspects of problem solving behavior, we desire a single theory that explains all of these abilities. In this paper, we outline five theoretical assumptions that we have adopted to this end. We then introduce an implemented *hierarchical problem solver*, HPS, that instantiates these tenets. We describe the representational structures and the processes that act upon them to produce problem-solving behavior. In each case, we note how these aspects of the system contribute to our aims. Next, we characterize the strategic knowledge and domain expertise that modulate the problem-solving process. After this, we present our experience with HPS and report experiments that substantiate our claims about the system's functionality. We conclude by discussing related architectures and outlining avenues for future work.

2. Behavioral Abilities and Theoretical Claims

Our work is motivated by a number of phenomena that human problem solvers exhibit. In this section, we outline these behavioral abilities and some theoretical assumptions that account for them. Our work is premised on five established findings:

- Humans can *solve problems that they have never encountered*. Provided that the problem is not excessively complex, they will generally find a solution that achieves their goals.
- People *have many strategies at their disposal* to solve such problems; these techniques are both very general and domain independent.
- In conjunction with these strategies, they also *utilize domain expertise* to make their search for solutions more tractable.
- Humans can *adapt their strategy* to fit the current situation; that is, if they find that their approach is ineffective, they can switch to a more appropriate technique during their search.
- If problem solving on a task is successful, they *learn by committing that experience to long-term memory*; if they then encounter a similar problem in the future, they can apply the learned strategy to solve that problem with less effort.

We believe that these phenomena are important for any full account of the behavior of human problem solvers, and all of them have contributed to the design of our theory. We elaborate on the last two abilities in our discussion of future work, but for this paper we have limited our focus to the first three aspects: the capacity to solve novel problems, to do so in a flexible manner, and to utilize domain expertise when it is available. To account for these phenomena, we adopt five theoretical claims. Taken together, they constitute our core theory of problem solving:

1. Plans are represented by problem trees in which each child is a subproblem of its parent. If all of its subproblems have solutions, then the parent problem is also solved.
2. The search for solutions is organized as a tree, each node of which represents an elaboration that augments and refines its parent. A node can have several children that denote alternative elaborations.
3. The process that traverses and generates these structures involves four stages — state selection, decomposition generation, failure checking, and success checking — each of which involves decisions that are integral to problem solving.
4. At each stage, domain-independent strategic knowledge produces behavior. Control schemes include various techniques for traversing the search tree and decomposing problems, as well as other schemes. These control schemes combine to produce a range of domain-independent problem-solving strategies.
5. Domain expertise — that is, knowledge about a particular area — influences the decomposition of problems into subproblems and operates in the same manner as domain operators. Even if it is not sufficient to completely solve a problem, this knowledge may still contribute to the problem-solving process.

The first two claims focus on representational issues, the third with basic processing, and the latter two with modulating basic operations.

Problem decomposition was first introduced to the AI community by Newell, Shaw, and Simon's (1960) work on the General Problem Solver, but few modern problem-solving systems employ this

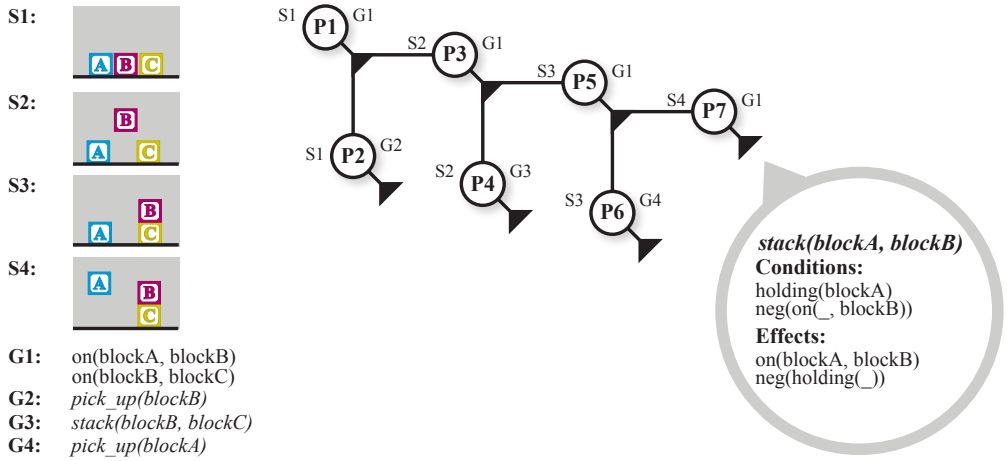

Figure 1. One possible solution for the problem P1 that describes four consecutive steps: pick up block B and stack on block C, then pick up block A and stack on block B.

idea. Our fourth postulate — providing support for a diverse range of domain-independent strategies — is similarly underused. In contrast, our second and fifth assumptions — using a tree to organize search and representing domain expertise as hierarchical methods — are both adapted by many AI researchers. We will discuss these comparisons further when we review related work.

We have incorporated these theoretical claims into a system, HPS, that is written in Prolog. In the sections that follow, we use examples from planning domains to illustrate our ideas and evaluate our system. However, our theory and system are capable of dealing with other varieties of problem solving, such as design, scheduling, and theorem proving.

3. Representation and Organization in HPS

Having established the abstract claims that constitute our theory of problem solving, we can now describe how the HPS system implements them. In this section, we focus on our first two claims, both of which are concerned with the system's principles of representation. We start by looking at the solutions HPS produces, and explain how their principal components relate to each other. We then clarify how these solutions are stored in a manner that supports flexible problem solving. In each case, we note why we need these structures to support flexible problem solving and hierarchical domain expertise.

3.1 The Structure of Problems and Solutions

Recall that our first theoretical postulate involves the representation of solutions — both partial and complete — as problem decomposition trees. These structures consist of two key components: problems and decompositions. These elements, and the relationships between them, are crucial to our account of both flexible problem solving and the integration of domain expertise.

We begin by looking at the *problem*, which represents the relationship between a *state description* and a *goal description*. Both descriptions can include a set of state elements, that is, predicates that encode facts about the state of the world, such as `on(blockA, table)`. In a state description, these elements represent the current state, but in a goal description, they represent some class of desired states. If a problem's state description matches all of its goals, then we call it a trivial problem. Goal descriptions may also include the identifiers of tasks that the problem solver must complete; such as `pick_up(blockA)`. These task descriptions typically correspond to domain operators that reside in a separate long-term memory. Each operator specifies *conditions*, *effects*, and *constraints*. The conditions encode the state elements that must be true for the operator to be applicable, while the effects describe the changes that will occur if the operator is applied. The constraints of the operator typically encode inequalities between the arguments of the operator, that will be adopted if the operator is used and enforced throughout problem solving.

The other key component of HPS's plans is the *decomposition*, which revolves around an instantiated domain operator. A decomposition describes how to partially or completely solve a particular problem. In many cases, it explains how to break the problem into two ordered subproblems: a *down subproblem* and a *right subproblem*. These problems are defined so that, when solved, they will act as a solution for their parent problem. Alternatively, a decomposition might simply encode an instantiation of a single operator without any subproblems; this *trivial decomposition* constitutes a solution to its parent problem. Decompositions in HPS are similar in structure to those used by GPS. However, as we discuss later, our system generates them in a very different manner.

To clarify the organization of these structures, consider the simple solution depicted in Figure 1. The initial problem, P1, belongs to a domain called the Blocks World (Fikes & Nilsson, 1972). In this domain, the agent must rearrange a set of blocks on a table so that they satisfy a specified configuration. In our example, block A is sitting on the table and blocks B and C are in a stack, with B on top. The initial goal description simply specifies that block A must be on top of block B.

The plan in the figure includes seven problems, the first, third and fifth of which are connected to decompositions that define two subproblems. P1's decomposition determines P2's goal description, which simply encodes the task of "picking up block A". There is a domain operator in long-term memory that, when instantiated, achieves this task; furthermore, all of its conditions match elements in P2's state description. Therefore, P2 may be connected to a trivial decomposition that instantiates this operator. The rest of the terminal nodes in the tree are also linked to trivial decompositions. The final problem, P7, does not have a task descriptor in its goal description, but it can still be associated with a decomposition that involves "stacking block A on block B". This is because the effects of that decomposition, if applied to P7's state description, satisfy all of P7's goals. This decomposition tree therefore represents a complete solution for P1; the plan involves four steps, all encoded by trivial decompositions: pick up block B and stack it on C, and then pick up block A and stack it on B. This example is simple, but it should illustrate the relations between components in a solution.

As we noted earlier, problems and their decompositions are fundamental components of HPS. However, this mode of representation alone is not sufficient to completely account for the flexibility exhibited by human problem solvers. For instance, it does not, support the encoding of multiple plans for a single problem or partial decompositions the system has abandoned. For this reason, we have incorporated an additional principle of representation, to which we now turn.

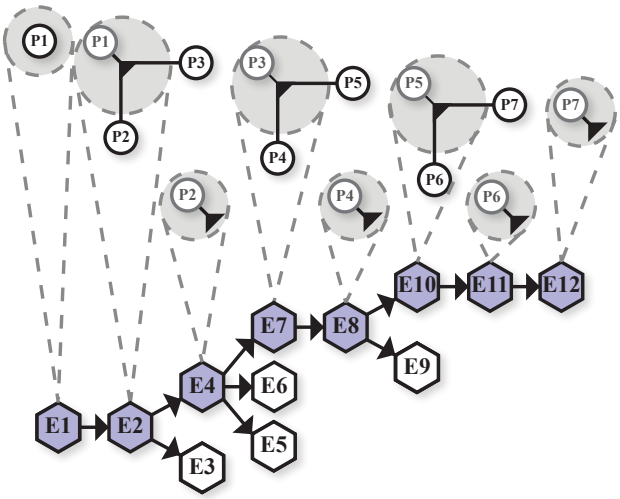

Figure 2. The lower graph represents the tree generated during search for candidate solutions. Each node contains a different elaboration of its parent, some of which map to the decompositions in Figure 1.

3.2 Organization of the Search Tree

Our second theoretical claim contends that the search for candidate solutions is organized as a tree whose edges denote elaborations. Each node in this connected space represents a plan, either partial or complete. The root node simply consists of the initial problem, but each subsequent node contains a new decomposition, including its conditions, goals, and any constraints that did not appear in its parent. If a decomposition has subproblems, then these too are stored in the node, along with their state and goal descriptions.

A node may have zero or more children, where each child represents an elaboration of its parent. For instance, if a node has three children, then each one will contain a different decomposition that elaborates on its parent's solution. Even though a node only explicitly stores the additions to the partial solution implied by its parent, it inherits all of the elaborations that lead from the root (the initial problem) to the current node. This encoding does not impede the system's flexibility and is far more efficient than storing an entire solution at each node. If every terminal node in its implicit plan has a trivial decomposition, then we say that node is *complete*.

To better explain how HPS uses this structure to represent the relations among plans, we will return to our Blocks World example. Figure 2 depicts a search tree for this problem that incorporates the solution from Figure 1. Each shaded node denotes a decomposition in our original solution. For example, E4 has three children. The last, E7, contains a decomposition that is involved in our solution — the decomposition (with subproblems) of P3. E4's other children, E5 and E6, encode alternative elaborations that result from using different instantiations of operators. Although each child only stores a single decomposition, they all inherit the elaborations of their ancestors. Therefore, they each denote a decomposition tree that consists of five problems.

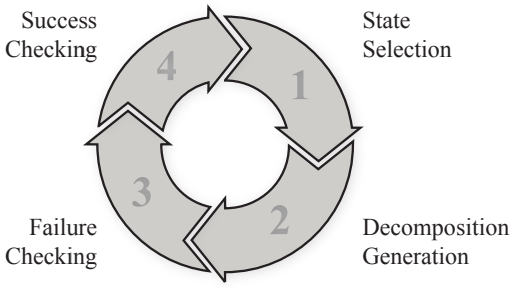

Figure 3. The four phases of the problem-solving process, as described in the text below.

This example should clarify how our two representational assumptions coexist and complement each other. Together, they provide the basis for all problem solving. In the following section, we describe how we have implemented our third claim, which is concerned with the problem-solving process that traverses and generates these structures.

4. Problem Solving in HPS

The structures described above play key roles in HPS's problem-solving process. At the outset, a single node contains the initial problem and its state and goal descriptions. The system recursively decomposes this problem and its subproblems, and, in doing so, adds new states to its search tree. It continues in this way until it finds a state that contains a solution to its initial problem. This process, which incorporates our third theoretical assumption, operates in a cycle that involves four stages: state selection, decomposition generation, failure checking, and success checking. We will consider each of these stages in turn.

Problem solving begins with the *state selection* stage, in which HPS chooses a node to concentrate on in the following stages. Only one is available at the outset — a node that contains the initial problem — but HPS may need to choose from a large pool of incomplete elaborations in later cycles. The selected state is annotated in working memory, as is the earliest unsolved problem in the selected solution. This selection remains active for the entire problem-solving cycle. In traversing these nodes, Suppose that FPS is in the process of solving the Blocks World problem in Figures 1 and 2 and that, in the previous cycle, it created E2's elaborations, E3 and E4. At the problem selection phase, FPS may now choose from E1, E2, E3 or E4. Assume that here it selects E4.

When it enters the *decomposition generation* phase, HPS may generate zero or more unique decompositions for its selected solution state, each of which it stores as an elaborated node in the search tree. It then adds these nodes to the list of available states. Let us return to our Blocks World example; at this stage, the system decomposes the only unsolved problem in E4's partial solution: P3 in Figure 1. The state description of this problem specifies that the gripper is holding block B, and this leads HPS to produce three alternative elaborations. Two of them involve different instantiations of the same operator — `stack(blockB, blockC)` and `stack(blockB, blockA)` — and the other uses `put _down(blockB)`. In each decomposition, the down subproblem's goal description refers to the task of using the instantiated domain operator, and the right subproblem's goal description includes all of its parent's goals.

Next, HPS shifts to *failure checking*, where it determines whether any of the newly created nodes in its search tree represent a failed plan. The system annotates any such nodes and removes them from the set of nodes available to the system during state selection. Suppose that, having selected E6, it created an elaboration that encoded the "put A on the table" domain operator. This operator produces a state that is identical to the initial state description — block A is back on the table, and block B and C are still in a pile. At this point, HPS might choose to mark this node as failed and remove it from the list of available elaborations.

In the final stage, *success checking*, the system ascertains whether it has found enough solutions. To do so, it considers the decomposition tree that the node implicitly represents, rather than the elaboration that it explicitly stores. Upon finding a suitable number of complete nodes, the system can terminate the problem-solving process. Imagine that, having marked E7 as failed, HPS re-selects E5 and generates the decomposition in E8. At the next success checking stage it finds that this node, which represents the entire decomposition tree in Figure 1, is a solution to the top-level problem; in this instance, it chooses to end the problem-solving process.

To summarize, problem solving in HPS consists of four stages. For each cycle, the system chooses a solution state to focus on, generates zero or more decompositions for it, decides if it should fail any those elaborations, and, finally, ascertains whether it has found enough complete solutions. This process is nondeterministic, and the choices made in our example constitute just one particular variety of problem solving. We noted earlier that our system utilizes the same decompositional structures as GPS. However, unlike that system, HPS is not restricted to using means-ends analysis to generate its subproblems. In the following section, we explain how the system supports alternative schemes at all four of its stages.

5. Modulating Problem-Solving Behavior

The structures and processes described in the previous sections provide a foundation for our system but by themselves do not account for the phenomena that we wish to explain. To emulate both the flexibility exhibited by human problem solvers and their ability to exploit domain expertise to make search more tractable, HPS incorporates knowledge to produce behavior. We begin by focusing on the strategic knowledge that produces domain-independent control schemes and then discuss domain expertise encoded as hierarchical structures.

5.1 Representing and Using Strategic Knowledge

When humans encounter a novel problem, they employ some strategy to organize their search for a solution. They might work forwards from their current state or backwards from their goal; they might consider several alternatives at once or follow one path until they either reach a solution or give up. Schemes such as these are not concerned with specific domain elements, so they can be adapted to almost any situation. The ability of human problem solvers to use such strategies is one of the important phenomena that we seek to explain. To account for this ability, we incorporate domain-independent strategic knowledge that produces alternative behavior at each stage of problem solving. This relates to the fourth theoretical postulate that we proposed in Section 2.

This knowledge comprises strategic control rules that reside in long-term memory. Each rule is ascribed to a particular stage of the problem-solving cycle, and encodes both a set of conditions and a set of actions. All of these elements are domain independent in that they refer to meta-level predicates, such as *problem*, *state*, *goal*, *decomposition*, *precondition*, and *effect*. For example, a rule might specify that "if a solution state exists such that its depth exceeds a particular limit, then mark that state as failed".

Strategic knowledge influences every stage of the problem-solving cycle. Upon entering a stage, the system begins to make its way through the relevant strategic knowledge. It instantiates the actions of the first applicable rule, and if the conditions match working memory in more than one way, it will choose a particular instantiation at random. After firing a single rule, the system returns to the beginning and starts again. A 'refraction' scheme prevents the same rule instance from firing again until the system returns to the stage in a subsequent cycle. Generally, once it has fired between zero and ten rules, HPS finds that there is no more applicable knowledge and moves on to the next stage. To support alternative control schemes, one adds and removes rules to and from the knowledge base.

To illustrate the affect that this knowledge can have on HPS's process, we consider just some of the control schemes available to each stage:

- Strategic knowledge for the first phase supports various search regimens. For instance, selecting nodes that were created recently will produce depth-first search or iterative sampling (Langley, 1992), while selecting older nodes will result in breadth-first search or beam search.
- At the second stage, decomposition generation, strategic knowledge might specify a domain-independent heuristic that HPS can use to evaluate candidate decompositions. For example, it may lead the system to rank decompositions according to the number of goals they would achieve or the conditions that are not yet satisfied. Strategic control rules also define the states and goals of decompositions' subproblems. By doing so, they support techniques like forward chaining, means-ends analysis (Newell, Shaw, & Simon, 1960), and full regression planning (McDermott, 1991).
- For the third stage, strategic knowledge defines criteria for failure. For instance, control rules might stipulate that the system should fail an elaboration if it produces a loop — that is, if the state that results from the decomposition's effects matches another state higher up in the plan. Alternatively, knowledge for this stage might cause HPS to fail a node that has reached some depth limit or for which the system cannot generate any more elaborations.
- Finally, the strategic knowledge for the fourth stage determines the system's criteria for success. These control schemes relate to both the completeness and the number of plans that HPS returns. For instance, strategic knowledge might induce HPS to return a solution that satisfies all of the initial goals or only a particular number or percentage of them. Additional control schemes might instruct the system to continue until it finds a specified number of solutions.

In summary, strategic control rules drastically alter HPS's behavior at each phase of the problem-solving process. By combining these rules in various ways we can reproduce a wide range of domain-independent strategies. Different approaches are suited to different domains and problems, so a flexible problem solver should support as many strategies as possible.

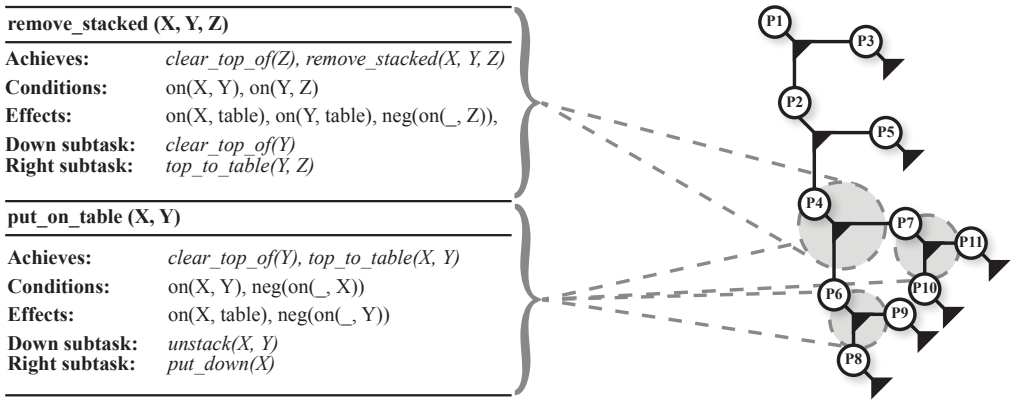

Figure 4. A solution to a Blocks World problem that involves both domain operators and domain expertise in the form of hierarchical methods.

5.2 Representing and Using Domain Expertise

The third phenomena that we wish to explain is humans' ability to bring domain expertise to bear during problem solving. Domain knowledge — even if it is only partial — can make the process much more tractable and is thus a desirable ability to reproduce. Recall that our fifth theoretical assumption is that one can utilize expertise alongside simple domain operators. This form of domain expertise is often represented by hierarchical task networks (HTNs) that consist of a set of methods, each of which defines a way of breaking some task into smaller subtasks, and we have taken a similar approach. Each method specifies the tasks that it achieves and two ordered subtasks. However, unlike those in traditional HTN planning systems such as SHOP2 (Nau et al., 2003), methods in HPS may also define the effects that they produce.

Our system draws upon domain expertise at the decomposition generation stage. It can retrieve a hierarchical method from long-term memory and instantiate it to decompose the selected problem. The method's conditions and effects become those of the decomposition, and its down and right subtasks determine the tasks and/or goals of the down and right subproblems. Regardless of whether HPS uses domain expertise or first-principles planning to decompose a problem, the resulting structure is the same. This reinforces our idea that problem solving in general is a process of hierarchical decomposition.

HPS's principles of representation and processing let it utilize methods in three key ways. If it is solving a problem that only specifies tasks, then the system will carry out traditional, task-driven HTN planning. Alternatively, when it comes across a problem that stipulates both tasks and goals, HPS only considers those that achieve the task *and* the goals. A third scenario involves problems that only specify goals. In this case, the system simply treats methods as it does domain operators. In addition to selecting those that are applicable in the current state, our system can also select ones that satisfy a certain number of goals, or meet other criteria that is specified by strategic knowledge.

An example solution from the Blocks World domain should serve to clarify how the system uses these methods. This problem is more complicated than our previous example; at the outset, the three blocks are in a single stack, with C at the bottom and B on top. The goal description specifies that

block B must be on top of C. Suppose that HPS adopts depth-first search with means-ends analysis, and that it ranks methods according to the number of goals that they achieve. As Figure 4 illustrates, the system's domain expertise consists of just two methods. The first, `remove_stacked(X, Y, Z)`, specifies `clear_top _of(Y)` as the down subtask, and `top_to_table(Y, Z)` as the right. The second method, `top_to_table(X, Y)`, has `unstack(X, Y)` and `put_down(X)` as its subtasks, both of which may be solved by domain operators. Note that the `top_to _table` task achieves both of the tasks specified by `removed_stacked`.

Since the initial problem does not specify a task that might solve it, a traditional HTN planner could not even attempt to solve it. HPS can still use domain expertise when the problem does not specify a task; but in this case, strategic knowledge stipulates means-ends analysis, and the system cannot instantiate either method in a way that achieves one or more of its initial goals. Instead it turns to first-principles planning and creates successive decompositions that involve the "stack block B on C" and "pick up B from the table" domain operators. The conditions of the latter method become the goals of P4. Since the `remove_stacked` method satisfies all of these state goals, HPS can instantiate it and use it to decompose P4. The subproblems of this decomposition take the method's subtasks as their goals. From here, HPS performs more traditional HTN planning by selecting methods that achieve these subtasks.[1] It is possible that domain operators might select domain operators to decompose these subproblems in alternative elaborations, but the `top_to_table` method provides a much more direct route, as the figure illustrates.

This example demonstrates that HPS's principles of representation and processing let it unify knowledge-rich and knowledge-lean problem solving. Domain expertise facilitates the process when it is available, but the system can still fall back on search when its knowledge is incomplete. In either case, the structures that HPS produces are the same, which reinforces our conception of problem solving as a process of hierarchical decomposition.

6. Experience with the HPS System

Now that we have described HPS, we can report our experience with the system. In Section 2, we outlined the phenomena that we wish to explain; namely, that humans can solve novel problems, they can use a diverse range of strategies to do so, and they can utilize domain expertise, when it is available, to make the problem-solving process more tractable. We begin this section by analyzing the results of experiments that relate to the first phenomenon. We then demonstrate HPS's coverage of problem-solving strategies before showcasing how it utilizes expertise to aid problem solving.

6.1 Basic Problem-Solving Ability

We focus first on our system's ability to solve novel problems. To test this, we have encoded the primitive operators for both the Blocks World and Logistics domains. The Blocks World is a stacking puzzle in which labeled blocks must be stacked on each other or on the table using operators for picking up a block from the table, putting it down, stacking and unstacking. The

1. In this example, the available domain expertise equates to the 'bottom part' of an HTN, but HPS could just as easily start with a method from the 'top part' of an incomplete network. After exhausting its expertise, it would simply fall back on first-principles planning to decompose any unsolved subproblems.

Logistics domain involves transport problems in which labeled objects must be moved between locations. Trucks can transport objects between locations within a city, while airplanes can transport them between the airports of different cities. This domain also includes operators for loading and unloading objects, driving trucks between locations, and flying an airplane between airports.

We provided HPS with the knowledge for each of these domains, along with the strategic control rules required to perform depth-first search with means-ends analysis and with termination upon finding a single solution. We created a set of ten problems for every domain and then ran HPS on each of them, with the problems loaded into the system's working memory. HPS successfully solved six of the ten Blocks World problems, but only three of those from the Logistics domain. All failures in problem solving were the result of reaching the specified limit, at which point the system gave up. While the performance of this simple search strategy leaves much to be desired, the runs demonstrate that HPS was able to solve novel problems from both of these domains as intended.

6.2 Supporting Flexible Problem Solving

Our second claim is that HPS can use a wide range of domain-independent strategies to solve problems. To test this hypothesis, we have combined strategic control rules for each of HPS's four stages to produce six distinct problem-solving strategies. Many of these strategies share control knowledge when two strategies require the same choices at a given stage. We chose these six approaches to represent a variety of techniques that have appeared in the AI literature. These included:

- *Forward breadth-first search*, which results from a combination of breadth-first state selection, generation of intentions through forward chaining, loop-triggered failure, and success upon finding a single state that matches the initial goal description.
- *Forward search with iterative sampling* (Langley, 1992), which combines iterative sampling search — repeated stochastic depth-first search with no memory of previous passes — to a fixed depth, forward chaining, loop-triggered failure, and success on finding a solution.
- *Depth-first means-ends analysis*, which results from a combination of depth-first state selection, intention generation through backward chaining, loop and depth triggered failure, and success upon finding a single full solution.
- *Forward best-first search*, which uses best-first state selection, forward chaining, and loop-triggered failure in which success is triggered after finding three distinct solutions. We also produced a variant of *forward best-first search* in which success was triggered after finding three distinct solutions, rather than after uncovering only one.
- *Three-step lookahead*, a strategy that results from forward search, look-ahead state selection, forward chaining, loop-triggered failure and success on finding one full solution.

We tested HPS on each problem that we had defined previously for the Blocks World and Logistics domains. We ran every combination of strategy and problem ten times to account for nondeterministic processing. If the system reached 500 cycles without finding the desired number of solutions, it terminated the problem-solving process and considered that run a failure. HPS successfully solved problems using each of the six strategies described, but performance on many problems was poor. Forward best-first search was the most successful strategy in our tests was forward best-first search, which consistently solved all of the tested Blocks World problems, but only able solved

four tasks from the Logistics domain. Forward breath-first search found solutions for five of the ten Blocks world problems, but it could not solve any of the Logistics problems within the cycle limit. This was probably because Logistics has a high branching factor, a domain characteristic that is well known to cause difficulties for breath-first search.

These experiments demonstrated HPS's ability to use a variety of domain-independent problem-solving strategies. Complex problems may pose a substantial challenge for such strategies, as can be seen in our results, but the ability to support them is a key component of any system that seeks to explain the main features of problem solving.

6.3 Problem Solving with Domain Expertise

Our third claim relates to use domain expertise to decrease the amount of search required to solve problems. As noted earlier, HPS can use hierarchical methods in a variety of ways, depending on the form of the problem it is trying to solve. To study the effects of domain expertise, we created domain-dependent methods for both Blocks World and Logistics. The former had four tasks for getting blocks in the gripper, assembling stacks of blocks, and dismantling them. The latter had ten methods for five distinct tasks for delivering within and between cities, as well as smaller delivery steps like dropping off a package once it is within a truck.

In our initial runs, we provided HPS with five problems from each domain with task-like goals, as is traditional in the HTN paradigm. The system solved every problem in both domains, which was unsurprising because HTN decomposition requires very little search and domain expertise was available to decompose each task. We also provided HPS with five problems in each domain whose goals were a mixture of state-like elements and task names. In these runs, only methods that both matched the problem's task *and* achieved the state-like goals could contribute to a problem's solution. As before, the system solved all five tasks in each domain. As task names limited branching factors, HPS encountered no difficulty in finding decompositions that satisfied the state-like goals.

Another study examined HPS's ability to benefit from domain expertise when given only state-like goals. Here we provided it with the knowledge required for forward chaining, depth-first search guided by a simple domain-independent heuristic. For both domains, we provided the same domain knowledge as before and ran it on the problems from Subsection 6.2. Using this expertise, HPS solved all ten of the problems, as the methods let HPS quickly see what operators were needed to achieve its goals. Although the system solved three problems by expanding only one method, other problems required HPS to chain methods and operators to achieve its goals. Although four methods was the longest chain used in any of these solutions, such a capability already exceeds that of traditional HTN planners.

In summary, our experiments supported the three core claims about HPS's abilities. Our demonstration that the system can solve novel problems, but that it also finds some difficult, is neither unusual or surprising, and we included it only for the sake of completeness. The second study is more important in its confirmation that HPS supports a variety of distinct problem-solving strategies in a compositional manner. Finally, the third set of runs showed that the system can use hierarchical domain knowledge to solve traditional HTN tasks, problems that include only goal descriptions, and mixtures of them, and that it can also combine partial hierarchical knowledge with primitive operators to make problem-solving tractable.

7. Related Research

At the outset, we presented five principles that comprise our theory of problem solving. In this section, we discuss previous work that is relevant to each of these ideas in turn, outlining what it shares with our framework and how it differs.

One claim is that *problem solutions are represented by decomposition trees*. This idea is widely adopted in work on theorem proving and HTN planning, which encode solutions as hierarchical decompositions based on domain-specific rules. Models of expert performance in geometry and physics incorporate a similar assumption. This differs from means-ends models of problem solving, which use domain-independent techniques to search a space of problem decompositions (Newell et al., 1960; Fikes & Nilsson, 1972; Carbonell et al., 1960; Li et al., 2012). This idea plays a central role in HPS, but our system separates it from a reliance on means-ends analysis and allows combination with other strategies. Marsella and Schmidt's (1993) PRL utilizes problem decompositions much like those in HPS, but encodes its structures as AND/OR trees that include disjunctive choices. Research on problem decomposition has received less attention in recent decades, but we believe it remains an important principle for general problem solving.

A second primary assumption is that *search is organized as a tree, in which each node is an elaboration of its parent*. Of course, search trees have been prevalent in AI research since the Newell et al.'s (1960) groundbreaking work. Searching through a space of incremental elaborations on partial solutions has been less common, but was central to work on refinement planning (Kambhampati et al., 1995). Within this paradigm, partial-order planners like SNLP (McAllester & Rosenblatt, 1991) searched for solutions by adding actions and ordering constraints to partial plans, starting with an empty structure. Our theory and implementation takes a similar approach, but instead uses decomposition to elaborate on previous solution structures.

Our third assumption is that *the problem-solving process involves a cycle with discrete stages*. All cognitive architectures operates in cycles, each with stages, but these typically involve matching, selection, and application of production rules, with no special attention given to problem solving. Two notable exceptions are Soar (Laird et al., 1987) and Prodigy (Veloso et al. 1995), both of which focus on problem solving and include staged processing. In Soar, these include elaborating the current problem state, selecting an operator to apply, and applying this operator. This contrasts with HPS's four-stage approach of selecting states, generating decompositions, checking for failure, and checking for success. Prodigy has stages for goal selection, operator selection, and variable binding; the architecture includes steps for operator generation and backtracking, but it cannot modulate them using strategic knowledge, as in HPS. Our system also differs from Prodigy in that it can use search techniques other than means-ends analysis; the FLECS (Veloso & Stone, 1995) extension also exhibits this ability, but HPS supports even greater flexibility.

The fourth assumption of our theory is that *domain-independent strategic knowledge modulates problem solving*. This idea has a long history in AI, but it has never been widely utilized. Laird's (1984) early work on Soar emphasized the importance of 'weak methods' in problem solving and used domain-independent control rules to reproduce a variety of search techniques. However, Soar makes weaker assumptions than HPS about solution representation, which lets our system state control rules at a higher, more abstract level that we believe users will find easier to specify and understand. Prodigy can also fall back on domain-independent knowledge when domain expertise

is unavailable, but papers on that architecture have not emphasized this ability and strategies are limited by its commitment to means-ends analysis.

Finally, our fifth assumption is that *hierarchical domain expertise influence the decomposition of problems into subproblems*. HTN planning systems constitute one extreme of this idea, in that they rely completely on domain knowledge to solve problems. Although HTN techniques support very efficient planning, they break down when knowledge is incomplete. This contrasts with HPS, which can utilize domain expertise when it is available but can combine it with first-principles search when it is only partial or even entirely absent. A systems have attempted to combine HTN-like planning with state-space planners. DUET (Gerevini et al., 2008) integrates HTN planning with a local first-principles search technique, but it cannot use domain expertise to decompose a problem into smaller ones with subgoals. On the other hand, GODEL (Shivashankar et al., 2013) only supports domain methods that create goal-oriented subproblems, preventing it from carrying out true HTN planning with subtasks. One can replicate HPS's approach to method decomposition in Soar, but this would produce a particular system implemented in that architecture, rather than making a theoretical commitment on how to encode and utilize domain expertise in problem solving.

Our previous system, FPS (Langley et al., 2013), attempted to implement some of the ideas presented in this paper. FPS supported a variety of problem-solving behaviors using strategic control rules similar to those in HPS, but its representation prevented strategies that produce multiple solutions to problems at different levels of decomposition. The system lacked a distinct state space, instead letting single problems have multiple decompositions and selecting problems at any level for further decomposition. Moreover, FPS could not take advantage of domain expertise, which we have demonstrated is a powerful way to decrease problem-solving effort. Thus, the theory embodied in HPS constitutes a clear advance over this earlier work.

Each of these earlier systems share at least one of HPS's theoretical assumptions, but none combines all of them in a single, integrated framework. Soar is general enough to be able to produce the same behavior as HPS, but it avoids architecture-level commitments about some issues that we believe contribute to a more complete theory of problem solving.

8. Concluding Remarks

At the beginning of this paper, we discussed several human problem-solving abilities, including the facts that people can solve novel problems, that they can do so in a flexible manner, and that they can utilize complete or partial domain expertise to make this process more tractable. In response, we proposed five theoretical assumptions that, together, can account for this behavior, and we then HPS, a hierarchical problem solver that incorporate these ideas. We discussed both the organization of solutions and the four-stage problem-solving cycle that generates them, then introduced the domain-independent strategic knowledge and domain expertise that HPS uses to produce behavior. To demonstrate that HPS has accomplished our initial goals, we presented three sets of experiments across a range of domains. We then discussed architectures that are similar in spirit to HPS.

We believe that our work to date constitutes substantial progress towards a general account human problem-solving behavior, but there remain several directions in which we should extend our theory and system. As noted in Section 2, we designed our framework with additional aspects of

problem solving in mind, including people's ability to adapt their strategy to fit different situations. Our system already supports a wide range of techniques that are governed by strategic knowledge, but we should extend it to switch between strategies as appropriate. This functionality would require making meta-information about the state of problem solving available to the system. Examples are information about the branching factors in the forward and backward directions or the number of attempts the system has made to elaborate on a particular node. We must also extend the HPS's strategic control rules to make them conditional on this meta-data. These two extensions should let our system effectively support adaptive problem solving.

We should also extend HPS so that it can learn from successful problem solving. The similarities between our system's decomposition trees and the structure of its methods provide the groundwork for this functionality. In fact, we hypothesize that hierarchical domain methods are simply the generalized traces of successful problem decompositions. This means HPS can learn by storing generalized versions of successful decompositions in long-term memory, using an analytical process could then determine conditions and effects for these new methods, so that they can be called upon in the future. This approach is similar to the one reported by Li, Stracuzzi, and Langley (2012) in their work on learning in ICARUS. Once we have extended the system in these two directions — supporting adaption and method learning — HPS will provide an even more flexible and inclusive account of problem solving.

Acknowledgements

This research was supported in part by Grant N00014-10-1-0487 from the Office of Naval Research. 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

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

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

Fikes, R. E., & Nilsson, N. J. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, *2*, 189–208.

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.

Kambhampati, S., Knoblock, C. A., & Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, *76*, 167-238.

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

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. Choi, D., & Rogers, S. (2009). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, *10*, 316–332.

Langley, P., Emery, M., Barley, M., & MacLellan, C. (2013). An architecture for flexible problem solving. *Poster Collection: The Second Annual Conference on Advances in Cognitive Systems* (pp. 93–110). 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.

McAllester, D., & Rosenblatt, D. (1991). Systematic nonlinear planning. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 634–639). Anaheim, CA: AAAI Press.

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 S. Minton (Ed.), *Machine learning methods for planning*. San Francisco, CA: Morgan Kaufmann.

Nau, D., Au, T., Hghami, O., Kuter, U., Murdock, J. W., 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. (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, China: AAAI Press.

Veloso, M., Carbonell, J., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *7*, 81–120.

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