Advances in Cognitive Systems 3 (2013) 1-18 Submitted 9/2013; published 12/2013

An Architecture for Flexible Problem Solving

Pat Langley PATRICK.W.LANGLEY@GMAIL.COM
Miranda Emery MEME011@AUCKLANDUNI.AC.NZ
Michael Barley MBAR098@CS.AUCKLAND.AC.NZ
Department of Computer Science, University of Auckland, Private Bag 92019, Auckland 1142, NZ

Christopher J. MacLellan CMACLELL@CS.CMU.EDU
Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract

The literature on problem solving in both humans and machines has revealed a diverse set of strategies that operate in different manners. In this paper, we review this great variety of techniques and propose a five-stage framework for problem solving that accounts for this variation in terms of differences in strategic knowledge used at each stage. We describe the framework and its implementation in some detail, including its encoding of problems and their solutions, its representation of domain-level and strategy-level knowledge, and its overall operation. We present evidence of the framework's generality and its ability to support many distinct problem-solving strategies, including one that is novel and interesting. We also report experiments that show the framework's potential for empirical comparisons of different techniques. We conclude by reviewing other work on flexible approaches to problem solving and considering some directions for future research.

1. Introduction

The ability to solve novel problems is one of the hallmarks of human intelligence, and thus a desirable feature of any artificial cognitive system. Not only are people able to construct innovative solutions to tasks they have never before encountered, but their behavior along these lines exhibits two important features. The first is that problem-solving abilities are very general, in that people apply them to a wide range of settings. The second is that humans exhibit substantial variation in their problem-solving strategies, both across different people and across distinct tasks. A full computational theory of problem solving should account for both of these characteristics.

Some early computational studies of problem solving (e.g., Newell et al., 1960) focused on generality, but this emphasis has become far less common in recent decades. Most systems are now specialized to certain classes of problems, such as planning, scheduling, and design, which has led to efficient implementations but also to capabilities that are less general than those found in humans. In addition, most implemented systems adopt a single problem-solving strategy, and in some cases a narrow class of methods dominates an entire subfield.

In this paper, we return to AI's initial concern with generality and address the additional challenge of variability. We present a theoretical framework that supports generalized problem solving and that accounts for variations in terms of differences in strategic knowledge. We argue that the theory provides new insights because it not only covers well-known strategies but also suggests novel ones that have not appeared in the literature. The framework also lets us compare the performance of different strategies in a controlled manner, and we illustrate this point with experimental studies that reveal some interesting regularities. In closing, we relate our approach to earlier frameworks – including Soar and PRODIGY – that have addressed variation in problem-solving strategies and then consider some avenues for future research. We note in advance that our aim is not to improve on these theories, but to complement them and explore facets of the topic they did not emphasize. Problem solving involves a broad enough class of phenomenoma that there seems room for multiple accounts, at least at this stage of our understanding.

2. Aims of the Research

We desire a computational theory of problem solving. A key step in developing any theory is to identify the phenomena one wants it to explain. In this case, the most basic fact is that people are often able to solve nontrivial problems that they have never before encountered. The realization that we might give computers the same ability served as one of the main foundations of the AI revolution in the 1950s, and it remains a major focus of the field. The central role of search in most AI courses and textbooks reflects this abiding concern.

A second phenomena is that the human ability to solve novel problems appears to be very general. Although experts take advantage of domain knowledge to handle familiar types of tasks effectively, they retain the ability to solve, with more effort, unfamiliar problems far outside their areas of expertise. Moreover, problem solving is not limited, as often assumed, to planning tasks that involve generation of action sequences. This ability is also useful in design, scheduling, theorem proving, and solving problems in mathematics and science. We would like a theory that supports such general capabilities.

A third phenomena, more underrated, serves as the primary focus of this paper: humans are highly variable in the strategies they employ to solve problems, and the AI literature on problem solving shows a similar diversity of approaches. For instance, we know that people sometimes chain backward from goals to select operators, as when solving physics problems (Larkin et al., 1980) and when working on puzzles like the Tower of Hanoi (Newell & Simon, 1972). In other cases, such as during chess play, they appear to chain forward from the current state (de Groot, 1978). Similar differences hold between early AI planning systems (Fikes & Nilsson, 1972), which chained backward, and more recent ones (Hoffmann, 2001), most of which chain forward.

Another dimension of variation involves the organization of search. Nearly every AI course teaches students about depth-first, breadth-first, and best-first methods, with the latter being popular in many implemented systems. Humans show less variability here due to their limited short-term memories, relying on progressive deepening (de Groot, 1978) and other variants of greedy search. Yet another issue, discussed primarily in the AI planning literature, concerns whether one engages in eager or delayed commitment of choices when selecting operators or bindings. There have been no studies of human biases on this front, yet it seems likely that they lean toward eager methods, due to memory limitations, but that they can delay some decisions.

A final dimension involves criteria for terminating search with either failure or success. Different problem-solving systems incorporate different conditions for abandoning branches, such as detecting loops and exceeding depth limits. They also adopt diverse schemes for determining success, ranging from the costly option of requiring all problem solutions to partial satisfaction with a single solution that achieves only some of the specified goals.

The computational account of problem solving that we develop should account for this variety of strategies, as well as be applicable to a broad range of task domains. Our purpose is not to construct a system that produces better solutions than others or that finds them more rapidly, but rather one that explains the full range of methods observed in human and machine problem solvers. We will not attempt to justify this goal further, as it seems desirable in its own right. However, we will note that such a framework would aid in experimental comparison of different strategies because it would enable their implementation in a common infrastructure, and we report initial studies along these lines in Section 4. In addition, such a theory seems a prerequisite for modeling the adaptive character of problem solving seen in humans, such as their ability to shift between forward and backward search on different problems, and we discuss ideas for extending our framework in this direction in Section 5.

We should note some other topics that are not a focus of our current research. One is that we have not attempted to account for all aspects of human problem solving, such as the influence of limited working memory on strategy choices, even though some clearly make different demands on memory than others. Another is that, following early work in the area, we have focused on problem solving on unfamiliar tasks in which little domain knowledge is available. A third is that we have chosen not to address domains that involve uncertainty. Each area is important, and we hope to explore them in future work, but we must limit our initial scope in order to make progress.

3. A Flexible Theory of Problem Solving

To reiterate, problem solving in humans and machines exhibits both considerable generality and great variety, and we want a theoretical framework that covers both of these features. In this section, we present such a framework, dividing our presentation into five parts. We begin by specifying the framework's theoretical assumptions, its representation of problems and solutions, and the architecture it employs to find solutions. After this, we characterize our formalism for encoding the domain knowledge over which the architecture operates. Finally, we describe how the framework encodes strategies that link the architecture with domain knowledge to solve specific problems.

Science often distinguishes between theories and models. In this view, our architectural framework constitutes a theory of problem solving that comprises principles of representation and processing. In contrast, domain knowledge and specific strategies combine to make up models that instantiate this theory to produce behavior. We have implemented both the theory and a variety of strategies in Prolog; we will refer to them collectively as FPS, which stands for *Flexible Problem Solver*, in homage to Newell, Shaw, and Simon's (1960) early work.

3.1 Theoretical Assumptions

Before we present our own theory, we should review the framework for problem solving that has become widely adopted in both AI and cognitive psychology. This standard theory is due originally to Newell, Shaw, and Simon (1958), but it has become so commonly accepted that few now question whether it has any potential for elaboration and improvement, which is the high-level objective of our research enterprise.

This theory states that problem solving involves carrying out search through a problem space in an effort to transform an initial state into one that satisfies a goal description. This problem space is not enumerated in advance, but rather is generated dynamically by applying operators that transform states into other states. These operators include both conditions on their application and effects they produce when applied. The search process may be guided by heuristics, but it is also organized by strategies that influence the order in which states are considered and in which operators are applied.

Our new theory of problem solving adopts all of these claims, but it also moves beyond them to incorporate some new postulates. These include assumptions that:

- The primary mental structure in problem solving is the *problem*, which includes a state description and a goal description.
- A problem *solution* consists of a problem P; an applied operator instance or *intention* I; a *down* subproblem that shares P's state but has goals based on I's conditions; a *right* subproblem that has the same goals as P but a state that results from applying I to P's state; and solutions to these subproblems. A trivial problem solution is one in which the state satisfies the goals.
- Problems and their (attempted) solutions reside in a *working memory* that changes rapidly over the course of problem solving, whereas operators and strategies reside in a *long-term memory* that changes gradually if at all.
- Problem solving operates in cycles that involve five stages: *problem selection*, *intention generation*, *subproblem generation*, *failure checking*, and *termination checking*. Each stage uses structures in long-term memory to produce changes to problem structures in working memory.
- Long-term memory contains two forms of content: *domain* knowledge that defines predicates and operators for the current problem domain and *strategic* knowledge that specifies the problem-solving strategies used at each of the five stages.

Although the first three assumptions specify important commitments about representation and organization, the final two tenets are the most interesting and important. We discuss them later in the section, after clarifying some issues of representation and organization.

3.2 Structure of Problems and Solutions

Before we discuss our framework's mechanisms for problem solving, we should first consider its representational assumptions. As noted, the primary structure is the *problem*, which always has an associated state description and a separate goal description, each with its own identifier so that it can be reused elsewhere. A nontrivial problem (one for which the state does not satisfy the goals) has zero or more operator instances or *intentions*, each of which decomposes it into a down subproblem and a right subproblem. A decomposition of problem P is a *solution* to P if its down subproblem and right subproblem themselves have solutions, with trivial subproblems serving as terminal nodes.

We can clarify this organization with an example from the Tower of Hanoi domain. Recall that this involves moving disks to target pegs subject to the constraints of moving only one disk at a time, not moving a disk if a smaller one is on it, and not moving a disk to a peg if a smaller one is already there. Figure 1 shows the structure of a solution for a simple task that involves moving disks 1 and 2 from peg B onto a target peg C that already holds disk 3. The simplicity of this problem

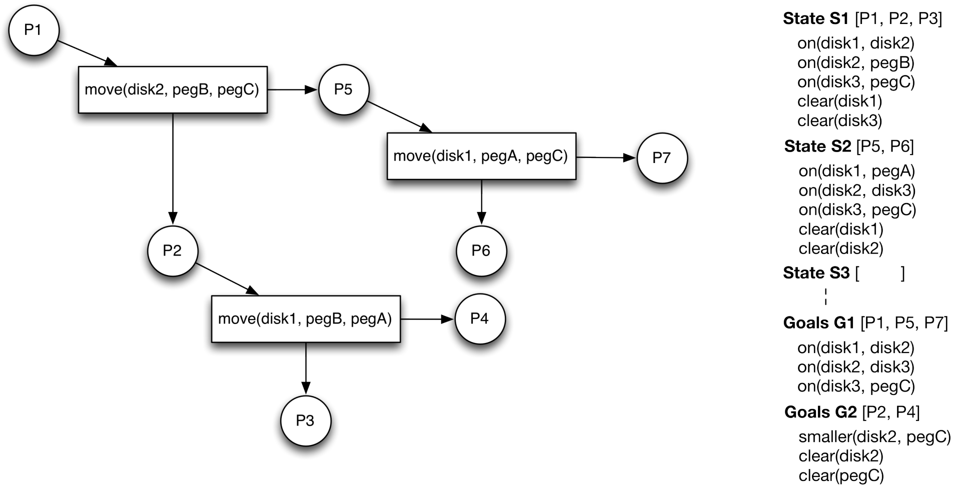


Figure 1. Solution decomposition for a simple Tower of Hanoi problem (P1) that includes both down-branching (P2) and right-branching (P5) subproblems. Each terminal node denotes a trivial subproblem.

(P1) is reflected in the structure of its solution. This is not the only possible solution structure for this problem, but it is a reasonable one that lets us clarify the organization of elements in memory.

This involves a decomposition of P1 into a down subproblem (P2), since its intention – moving disk 2 from peg B to peg C – is not applicable in state S1, and a right subproblem (P5), since this operator instance does not by itself achieve all of P1's goals (G1). The down subproblem P2 is decomposed further based on another intention – moving disk 1 from peg B to peg A – that produces two additional subproblems (P3 and P4). These are both trivial, in that the intention is applicable in S1 and achieves all of the goals (G2) associated with P2. The conditions for the original intention (moving disk B) match the resulting state S2, with a new state (S3) produced by its application that is associated with right subproblem P5. The intention attached to P5 – moving disk 1 from peg A to peg C – has trivial subproblems, as its conditions match S3 and it achieves the remaining goal in G1, thus solving the top-level problem.

This example is simple, but it should clarify the nature of problem decompositions and the organization of solutions, which is inherently hierarchical. We maintain that other frameworks are special cases of this general idea. For instance, the popular class of methods that approach problem solving in terms of forward search are techniques that only consider solutions with nontrivial right-branching decompositions. In contrast, the early Logic Theory Machine (Newell, Shaw, & Simon, 1957) only considered solutions with nontrivial down-branching decompositions. Our framework's support for both types of solution borrows from the decompositional scheme introduced in Newell, Shaw, and Simon's (1960) General Problem Solver. Although their implementation had many drawbacks, we hold that its representation of problem decompositions is ideally suited to support the variety of search strategies that we aim to explain.[1]

1. The related technique of *island search* (Chakrabarti, Ghose, & Desarkar, 1986) also decomposes a problem into subproblems, but our approach comes closer to that in Newell et al.'s earlier work.

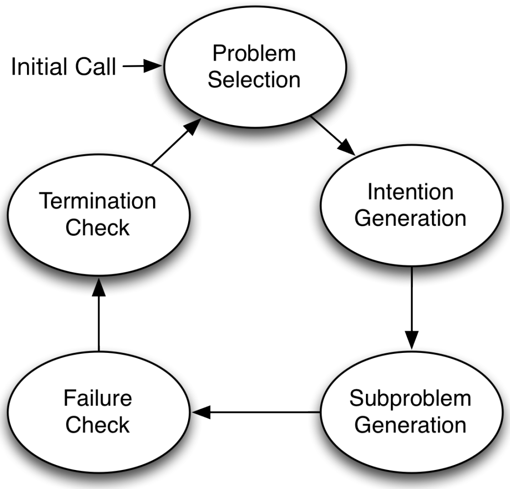


Figure 2. The five-stage cycle of the FPS problem-solving architecture.

3.3 Five Stages of Problem Solving

As just noted, our problem-solving architecture cycles repeatedly through five stages. The first step involves selecting a problem on which to focus attention. There is only one problem at the outset, but the number of choices increases on later rounds. During this stage, the problem solver examines working memory to determine which problems are available (i.e., not marked as solved or failed) and alters it to reflect the newly selected focus. Different strategic knowledge for this stage imposes different organizations on search through the problem space. For example, selecting more recently created problems leads to strategies like depth-first search and iterative deepening (Langley, 1992), whereas selecting less recent ones produces ones like breadth-first and beam search.

Once the architecture has selected a problem P, the second stage selects an operator instance that it hopes will lead to a solution for P. We refer to these as *intentions* because the problem solver intends them for execution if they participate in its final solution for P. This stage examines both P and domain knowledge about operators, considers all intentions that are available and relevant, selects one of them, and associates it with P. If no operator instances are available (e.g., if all relevant candidates have failed on earlier attempts), the problem solver annotates the problem to that effect. Different strategic knowledge for this stage generates intentions by chaining backward from problem goals,chaining forward from the state, or some mixture of these schemes.

The third stage generates new subproblems of the current problem P, if necessary, based on the newly selected intention I, if one has been produced. This includes creating a down subproblem that has the same state as P but has goals based on I's conditions, along with a right subproblem that has the same goals as P but has a state produced by applying I's effects to P's state.[2] Different strategies at this stage include eager commitment, which creates subproblems as soon as an intention

2. The resulting down subproblem will be trivial if I's conditions already match the current state, while the new right subproblem will be trivial if the result satisfies all of P's goals.

Table 1. Example FPS encodings for a state description, a goal description, and an operator for the Tower of Hanoi puzzle, with each being desribed as a set of literals.

State description:	*Operator description:*
state(s1, on(disk1, disk2)), state(s1, on(disk2, disk3)),	operator(move(Disk, From, To)),
state(s1, on(disk3, pegA)), state(s1, clear(disk1)),	condition(move(Disk, From, To), smaller(Disk, To),
state(s1, clear(pegB)), state(s1, clear(pegC)).	condition(move(Disk, From, To), clear(Disk)),
Goal description:	condition(move(Disk, From, To), clear(To)),
goal(g1, on(disk1, Any_disk)), goal(g1, on(disk3, pegC)),	effect(move(Disk, From, To), on(Disk, To)),
goal(g1, on(Any_disk, disk3)).	effect(move(Disk, From, To), clear(From)).

is available, and delayed commitment, which waits until a set of intentions have been found, then determines the order in which they should be applied.

Another stage checks for failures that indicate the problem solver should abandon the current problem *P*. This involves inspecting *P* and its associated intentions for unresolvable issues and adding relevant annotations to *P* about them. For example, this stage can check for an absence of available intentions, loops in the path that led to *P*, evidence that *P* is more difficult than one of its ancestor problems, reaching a depth limit, or even that too many attempts have been made to solve it. Such tests are not usually given the status of problem-solving strategies, but they are just as important as constructive steps that create intentions and subproblems.

The fifth and final stage checks to see whether any additional work is needed to solve the current problem *P*. This compares *P*'s state and goal descriptions to determine whether there is a sufficient match. If so, then it annotates *P* as solved, which will influence problem selection on the next cycle. Different strategic knowledge for this stage specifies alternative criteria for deciding when a problem is solved. This may require that all goal elements are matched, that some percentage of goals are satisfied, or, in settings where goals have associated values, that the summed value of matched goals exceeds a threshold.

To summarize, our problem-solving framework incorporates five stages: problem selection, intention generation, subproblem creation, failure checking, and termination checking. The architecture cycles through these steps, combining strategic knowledge associated with each stage and domain knowledge about operators to carry out search through the problem space that they jointly define. To understand how each stage operates, we must examine these knowledge elements, to which we now turn our attention.

3.4 Domain Knowledge for Problem Solving

According to the standard theory, problem solving depends on domain operators to generate new states that achieve goal descriptions, which in turn requires some way to represent all three types of structures. Our implementation of the FPS architecture adopts a logic-like notation for states and goal descriptions, in which each is specified by a set of literals (domain predicates with arguments). Goal descriptions may differ from states by omitting some elements, using variables rather than constants for some arguments, and including negated literals. Our notation for operators differs from traditional ones like STRIPS and PDDL, encoding each operator as a *set* of generalized literals, which supports more flexible processing by the problem-solving architecture.[3]

Table 1 shows the representation for a state description and a goal description from the Tower of Hanoi, along with that for the single operator for this domain. Each state literal describes a different aspect of the state, while the same holds for the more abstract goal description, which contains the pattern-match variable *Any_disk* in two elements. The operator description includes a literal that names the operator and its arguments, three condition elements that together specify when the operator is applicable and how their arguments relate, and two effect elements that jointly specify how the operator changes the state description. Although this example does not illustrate them, conditions may be negated and effects may involve deletions.

3.5 Strategic Knowledge for Problem Solving

Of course, domain knowledge by itself is not sufficient; we need some way to interpret the domain states, goal descriptions, and operators to produce problem solving. In most AI research, this takes the form of opaque, procedural code that, although efficient, lacks the flexibility needed to support a variety of strategies.[4] Instead, our framework specifies problem-solving behavior in terms of domain-independent knowledge.

The details of this strategic knowledge differ for each of the architecture's five stages, but they share some features that distinguish them from domain knowledge. In particular, they:

- make no reference to either domain-level predicates or operators, containing instead variables that match against such domain content;
- refer to meta-level predicates like *problem*, *state*, *goal*, *operator*, *condition*, *effect*, and *intention*;
- consist mainly of control rules that specify decisions to make under various conditions; and
- include inference rules that support control rules by determining whether their conditions hold.

The control rules for a given stage S are stored in an ordered list that, upon entering S, the architecture considers in turn, invoking inference rules as needed to determine whether a given control rule's conditions are satisfied. If so, then the rule fires and alters the contents of working memory. We can clarify this process by discussing briefly the control schemes that we have implemented within the framework.

Recall that the first stage involves selecting a problem on which to work. Here we have implemented three alternative regimes: depth-first search (two control rules), iterative sampling (two rules), and breadth-first search (two rules). Table 2 (a) presents the control rules used to produce depth-first search. Iterative sampling, which carries out repeated greedy search, replaces one of these rules but uses the other one, which provides evidence for the modularity of this knowledge.

The second stage is responsible for generating intentions (operator instances) for use on the current problem. In this case, we have implemented two approaches: backward chaining, which considers an operator only if its application would achieve one or more problem goals, and forward

3. We are not claiming that our formalism has greater expressive powr than traditional ones, only that its distributed character offers benefits for encoding and processing strategic knowledge.

4. We do not mean that procedural encodings cannot, in principle, offer flexibility, say through parameters or switches, but they have been associated empirically with inflexible approaches to problem solving.

Table 2. Control rules used to implement (a) depth-first search and (b) backward chaining.

(a) Depth-first selection of problems

If no problem is currently selected, then select the root as the current problem.

If the current problem has at least one descendent problem that has not been solved and has not failed, then choose one such descendent D at random and select D as the current problem.

If the current problem P has been solved or has failed and P has an ancestor A that has not been solved and has not failed, then select ancestor A as the current problem.

If the current problem P has not been solved and has not failed, then retain P as the current problem.

(b) Backward-chaining generation of intentions

If the current problem P has already been labelled as having no operators available, then do nothing.

If an operator instance I exists that achieves one or more goals for current problem P, and I is not already an intention for problem P, then make I an intention of P.

chaining, which considers an operator only if all its conditions match the problem state. Table 2 (b) presents the two control rules used for backward chaining; the forward version uses the same number of rules. The architecture also ranks intentions generated through both mechanisms by the number of goals they achieve. Informal studies suggested that these variants guided search substantially better than ranking candidates equally.

The next stage selects the most highly ranked intention, determines whether to proceed with this choice, and, if so, generates its associated down and right subproblems.[5] Here we have implemented two alternatives that involve eager and delayed commitment. The former utilizes four control rules, whereas the latter requires twelve. Delayed commitment also includes a substantial number of inference rules that detect interactions and dependencies (such as goal clobbering) among the candidate intentions and thus modulate their final ordering.

The framework's final two stages handle failure and termination. Control knowledge implemented for failure comprises nine rules, including ones that match when an ancestor problem is the same or more difficult than the current one, when all decompositions have been attempted, and when a depth limit has been exceeded. Analogous control knowledge for termination includes four rules that apply when the problem's state matches its goal description, when a right subproblem has been labeled as solved, when attempts to solve the top-level problem have failed, and when this problem has been solved.

Taken together, the control and inference rules for each stage specify a complete problem-solving strategy. Combined with the architecture, which cycles repeatedly through the stages, and the domain operators and predicates, they let the system attempt to solve novel tasks through a process of problem-space search. Each stage can utilize alternative schemes to organize and direct this search. Differences in control knowledge for each stage produces the variability we have argued is characteristic of human problem solving and desirable in cognitive systems generally.

5. When two or more intentions have the same ranking, the architecture selects among the candidates at random.

4. Experience with the Framework

Now that we have described our problem-solving framework and its implementation, we can turn to whether it satisfies the aims we outlined early in the paper. In this section, we examine the architecture's functionality, its generality, and its ability to support a wide variety of strategies, one of which is novel and interesting. We also report experiments that demonstrate the framework's support for empirical studies of alternative problem-solving techniques.

4.1 Basic Functionality and Generality

The most basic claims about our framework are that it supports effective problem solving on novel tasks and that it exhibits generality by solving problems across a wide range of domains. To demonstrate that it satisfies these criteria, we have implemented in Prolog (Clocksin & Mellish, 1981) the problem-solving architecture, a variety of problem-solving strategies that we will discuss shortly, and domain knowledge for various domains.

Table 3 lists the nine different domains for which we have developed predicates and operators. These include classic puzzles like Tiles and Squares (Ohlsson, 1982), the Tower of Hanoi (Newell & Simon, 1972), Missionaries and Cannibals, Slide Jump, and the Blocks World (Fikes & Nilsson, 1972), planning domains like Gripper, Logistics, and Dock Workers, and inference tasks like deducing kinship relations. These differ substantially in their characteristics. For instance, the three puzzles involve simple movement or stacking of objects subject to constraints, whereas the three planning domains incorporate more complex forms of action. These domains all involve nonmonotonic operators that can make existing relations false, whereas those for inferring kinship relations are entirely monotonic.

The fact that FPS can solve problems in each of these domains does not prove its generality, but it certainly lends evidence to that claim. However, many previous implementations, starting with Newell, Shaw, and Simon's GPS and continuing through many planning systems, have shown similar generality and have been tested more thoroughly. Thus, we now shift attention to our framework's more distinctive capabilities.

4.2 Coverage of Existing and Novel Strategies

Another primary claim is that our theoretical framework has broad coverage of the problem-solving strategies exhibited by both human and artificial problem solvers. By coverage, we mean that the architecture is capable of reproducing a wide range of strategies from the literature.[6] For example, if the framework could only reproduce the behavior of a single system, its coverage would be very poor, but if it could reproduce the behavior of a number of diverse systems, then its coverage would be high. This metric is both relevant and important because it shows how effectively our theoretical framework explains the phenomena in question, in this case the variety of problem-solving strategies. To evaluate the coverage of our system, we will describe how it reproduces several

6. Naturally, the size of this space is difficult to quantify in advance, but we will be able to tell, as our theory progresses, whether its coverage of known methods is increasing.

Table 3. Problem-solving domains on which we have tested the FPS architecture.

Tiles and Squares. This domain involves a set of N tiles that sit on $N + 1$ squares, so that one square is always empty. The single operator can move any tile from its current square onto the empty square, with the goal being to rearrange the initial configuration of tiles into a target arrangement.

Blocks World. This domain consists of a number of separate blocks and a table. The four operators let one pick up a block from the table or off another block and deposit a block onto the table or another block. Goal descriptions involve partially specified configurations of block and table relations.

Tower of Hanoi. This domain involves N disks that sit on three pegs. The single operator can move a disk onto a new peg if there is no smaller disk on it and if there is no smaller disk on the new peg. Goal descriptions specify desired placements of disks on pegs.

Missionaries and Cannibals. This domain concerns a boat, two river banks, N missionaries, and N cannibals. All missionaries and cannibals begin on one bank and must be transported to the other side, but there must never be more cannibals than missionaries on either bank. The single operator moves the boat from one bank to another, along with one to three passengers.

Slide Jump. In this domain, N dimes and N nickels are arranged in a row of $2N + 1$ locations, one of which is empty. Initially, all dimes are left of the empty slot and all nickels are right of it. One operator slides a dime right into an adjacent empty slot or slides a nickel left into one; the other jumps a dime right over a nickel into an empty slot or jumps a nickel left over a dime. The goal is to get all dimes to the right of the empty slot and all nickels to its left.

Kinship Relations. This domain involves people who are in parent-child relationships, with monotonic operators for inferring more complicated forms of kinship that appear in goal descriptions.

Gripper. This domain involves a robot with a gripper, a number of balls, and two or more rooms. Operators include gripping a ball, moving to a different room, and dropping a ball, with goals specifying desired locations of balls.

Logistics. This domains includes a number of cities, locations in those cities, transportable packages, and trucks and plans that can hold those packages. Operators involve loading and unloading packages, flying planes between cities, and driving trucks between locations. Goals specify desired locations of packages.

Dock Workers. This domain contains a robot, containers, pallets, and cranes in various locations. Operators let the robot move between locations and let the crane transfer a container from the top of a stack to the robot or vice versa. Goal descriptions specify desired positions of containers on pallets in locations.

problem-solving strategies from the literature and show how strategies that are not supported by our current models are still supported by the theoretical framework.

We have implemented some 12 distinctive problem-solving strategies that result from different choices of control knowledge for the five stages. These include some familiar techniques that are often presented as standalone problem-solving methods in textbooks:

- Breadth-first forward search. This strategy results from a combination of breadth-first problem selection, generation of intentions by forward chaining, eager commitment, loop-triggered failure, and success upon finding a single state that matches the goal description. Because this strategy only considers operators with matched conditions, it produces entirely right-branching paths that fan out from the generated states.

- Depth-first means-ends analysis. This strategy (Newell, Shaw, & Simon, 1960; Fikes & Nilsson, 1972) results from a combination of depth-first problem selection, generation of intentions by backward chaining, eager commitment, loop-triggered failure, and success upon finding a single state that matches the goal description. Because this strategy considers operators that achieve one or more goals but may not have the conditions satisfied, it typically generates solution paths with a mixture of down and right subproblems.
- Forward search with iterative sampling. This strategy (Langley, 1992) combines iterative sampling – repeated greedy search with no memory of previous passes – to a fixed depth with forward chaining to generate intentions, including eager commitment and success upon finding a single solution. This is another strategy that only considers operators with matched conditions, so it also generates paths with only right subproblems.
- Depth-first partial-order planning. This strategy combines depth-first problem selection with backward generation of intentions and delayed commitment for subproblem generation. Although this scheme typically produces solutions with a mixture of down and right subproblems, it checks for order dependencies among intentions before selecting one, applying it mentally, and creating right subproblems.

We have focused on these examples because they have been described in the literature and they will be familiar to many readers. However, a more interesting question is whether the framework suggests novel and interesting combinations of settings that have not appeared to date.

An important feature of our framework is that its modularity lets us combine any substrategies for one stage with any substrategies for another. This is what lets us produce so many distinct problem-solving strategies given the elements discussed in Section 3. Many of these are minor variations of each other, such as a depth-first version of forward search and a breadth-first variant of partial-order planning, while others correspond to techniques that have appeared in the literature but are not well known, such as the union of backward chaining with iterative sampling (Jones & Langley, 2005).

However, at least one strategy that emerges in this manner, and which is unexpected and interesting, involves a combination of least commitment with forward chaining. Least commitment was originally designed for use with backward-chaining methods, specifically ones for partial-order planning, but our framework also allows its use in conjunction with forward search. In this strategy, the problem solver generates a number of intentions with matched conditions, but then examines them for interactions before selecting one to drive creation of subproblems. To our knowledge, this idea has never been reported in the AI or cognitive psychology literature and, as we will see shortly, this scheme produces interesting and unexpected behavior.

This result provides evidence that our problem-solving framework does more than cover already established strategies; it suggests novel combinations that have never been observed. In this sense, it is playing a similar role to Mendeleev's periodic chart, which not only organized known elements into a coherent and structured framework, but also suggested new elements that would fill gaps in the table. The fact that our theoretical framework and its associated implementation supports such generality is an encouraging sign, although it seems clear that some strategies remain outside its current incarnation and there remains considerable room for improvement.

4.3 Experimental Studies of Alternative Strategies

One advantage of our framework's implementation is that it lets us compare the effectiveness of different strategies experimentally. Most modern problem-solving systems rely on a single, highly optimized strategy, so that differences in behavior may be due either to implementation details or to more basic distinctions. Because instances of FPS rely on the same underlying architecture and differ only in their strategic control rules, we can eliminate variance due to other factors and carry out more direct comparisons.

Figure 3 (a) presents experimental results on 17 distinct problems taken from five of the domains described earlier: the Blocks World, Tower of Hanoi, Kinship, Logistics, and Dock Workers. The graph plots the CPU times for all six of the forward-chaining FPS variants against all six backward-chaining versions. Each point shows the performance on a given problem of one forward-chaining strategy vs. its backward-chaining analog, holding settings for other FPS stages (e.g., problem selection or subproblem generation) constant. For problems above the diagonal, forward chaining generation of intentions took longer than backward chaining to find a solution, with the reverse holding for problems below this line.[7] One clear trend is that backward-chaining strategies fare better on the kinship domain; this result is unsurprising, as the operators are monotonic and thus lend themselves to goal-directed processing.

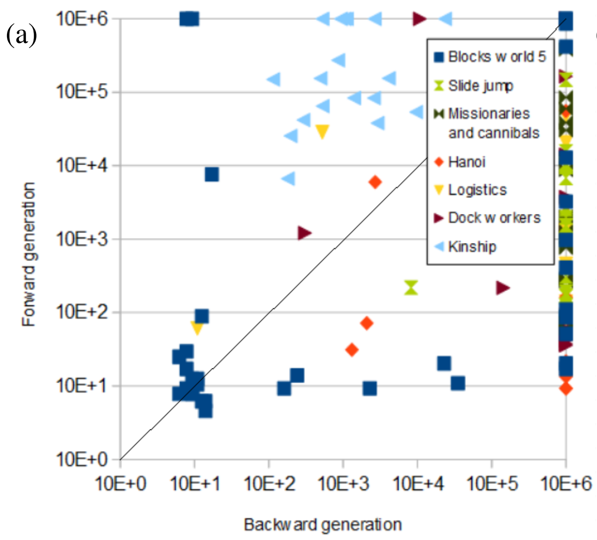
Figure 3 (b) presents the results from a more focused comparison between the novel class of strategies described earlier – the combination of forward chaining with delayed commitment – with the average of all other strategies. The graph shows that, except for some tasks in the upper right corner that are difficult in general, the first class of strategies fares better than the average of all other methods. Detailed analysis suggests that this combination is effective because it avoids the cost of checking for interactions on down subproblems, since these are trivial with forward chaining, and reaps the benefits of delayed commitment when it comes close to achieving the goal description, when interactions among operators often become important, without paying much overhead early in the search process.

These experimental results are intriguing and suggest the need for additional study, but they are not, in themselves, the main point of our research. The conclusion that readers should draw is that this comparison would not have occurred without our framework's capacity for suggesting and supporting novel problem-solving strategies, as well as its ability to compare such search methods experimentally in a controlled manner.

5. Related Research

Problem solving has played an important role in AI's history, and there is a substantial body of work from which we have drawn. Although early efforts (e.g., Newell et al., 1960) emphasized generality more than recent ones, even today's specialized planning systems exhibit substantial coverage of many tasks. Thus, we will not claim that our framework's generality is especially novel, despite its applicability to non-planning problems, such as inference tasks that involve monotonic operators.

7. We halted runs after 3,000 problem-solving cycles, so the cluster of points at the far right means that backward-chaining techniques exceeded this bound on many tasks.



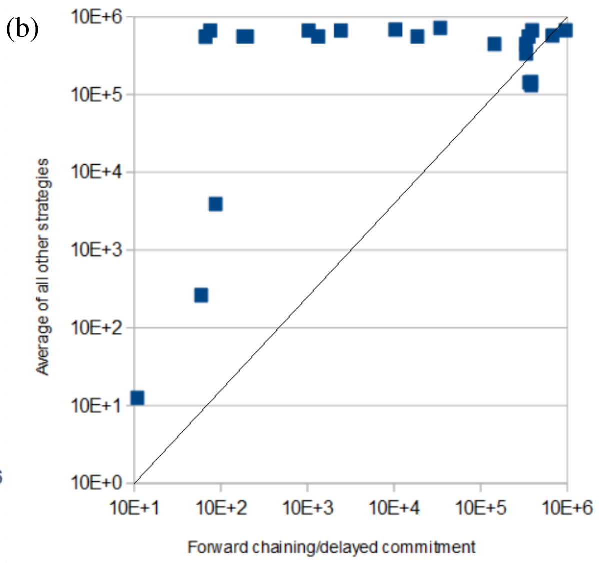


Figure 3. Scatter plots that compare the CPU seconds needed for FPS to solve problems in five domains (a) comparing variations on forward chaining with variants of backward chaining, as well as (b) comparing forward chaining, delayed commitment variants against averages of all other strategies.

However, the issue of variability is another matter entirely, as most research paradigms are now dominated by a single approach, such as forward search in planning and backward search in deductive inference. To find exceptions to this trend, we must examine the older literature, before the modern emphasis on systems that are optimized for CPU time. One example comes from Kambhampati and Srivastava (1995), whose framework for "universal classical planning" attempted to unify "plan-space" and "state-space" planning by casting each as special cases of refining a partial order plan. Their framework supported three distinct strategies: forward search through a state space, backward search through such a space, and mixed search through a plan space. Although similar in spirit, our framework provides much broader coverage of strategies.

Another early effort that supported a broad range of problem-solving behaviors revolved around the PRODIGY architecture (Carbonell, Knoblock, & Minton, 1990). This framework is quite similar to ours in that it incorporated a decision-making loop that, sequentially, selects among goals, states, operators, and bindings. These processing steps do not map precisely onto our five stages, but they play a similar role. PRODIGY also specified alternative strategies in terms of control rules, although an important difference is that published work on this framework emphasized using and acquiring domain-specific control knowledge, whereas we are concerned with domain-independent strategies. Minton (personal communication, 2012) reports that PRODIGY could utilize domain-independent control rules, but the project did not explore this ability systematically.

The FLECS system (Veloso & Stone, 1995), which extended the PRODIGY framework, came even closer to our approach by supporting both eager and delayed commitment, a capability handled by our third stage. FLECS could also shift between progression and regression approaches to planning, which maps directly onto alternatives in our architecture's second stage of intention generation. However, our framework also supports different schemes for problem selection, failure, and termination. Also, like PRODIGY, it only considered operators that would achieve at least one of the current problem's goals, so it could not produce purely forward search. Thus, we can view our FPS framework as an extended, more flexible version of FLECS that covers a wider range of problem-solving behaviors.

A fourth theoretical framework with similar aims is the Soar architecture (Laird et al., 1987). This organizes behavior around problem-space search in a six-stage cycle that checks for success, decides on failure or suspension, selects a state, selects an operator, applies the operator, and decides whether to save the resulting state. Each stage examines control rules that vote for or against available alternatives, complemented by an elaboration process that applies inference rules to inform these decisions. Although recent research on Soar has emphasized domain-specific control knowledge, early work utilized this approach to mimic a variety of "weak methods" that embodied generic problem-solving techniques. These correspond to our settings for various stages, such as depth-first search and backward chaining, and they could be combined in much the same way as FPS does. Our framework supports dimensions of variation not (to our knowledge) studied in Soar, such as eager and delayed commitment, but the two theories still have much in common.

These earlier efforts may lead some readers to question why we have developed our own framework for flexible problem solving. One reason is that, despite the early PRODIGY and Soar results, the topic has received remarkably little attention for over two decades, and it deserves more air time. Another is that FPS includes some new emphases, such as the hierarchical character of problem solutions and additional dimensions of variation. We will not claim that our theory is superior to its predecessors, but we believe that the field's understanding of problem is still far from complete, and that achieving human-level flexibility in this arena is challenging enough to justify more alternative accounts of this important set of phenomena.

6. Limitations and Responses

Despite our progress to date, there remain many directions in which we can extend both our theoretical framework and its implementation. The first step should be to support an even wider range of strategies in FPS by adding new control knowledge for various stages, especially problem selection, which should include techniques like beam search, best-first search, and iterative deeping. We should also add the ability to utilize numeric evaluation functions, both generic and domain-specific ones, to guide selection of both problems and intentions.

In addition, we should demonstrate that the framework supports other classes of problem-solving tasks. For instance, more flexible termination criteria should produce techniques for partial satisfaction planning (van den Briel et al., 2004), which find plans that achieve only a subset of the specified goals. The framework should also handle optimization techniques that do not halt at one solution but continue looking for alternatives that fare better on some evaluation function. Constraint satisfaction tasks should also be straightforward, requiring only that some goals be stated in terms of derived predicates. Naturally, we should continue to look for opportunities in which the fine-grained character of our control rules suggest novel strategies that support viable problem solving, in which case we should compare them experimentally to established techniques.

Of course, we should also explore extensions to the problem-solving architecture itself. One high priority is to support the use of hierarchical task networks ((Nau et al., 2001) to organize and constrain the search process. The decompositions that FPS already uses during during problem solving are similar in spirit to HTN methods. We intend to store generalized versions of these decompositions, with associated conditions and effects, as high-level operators on which the system can draw during search. A heuristic that prefers intentions which achieve more goals should favor hierarchical methods over primitive operators, which would still be available as fallback options. The mapping between specific problem decompositions and HTN methods also suggests the system can learn the latter from successful problem solutions.

Yet another promising avenue involves shifting between strategies dynamically, based on measures like relative branching factors in the forward and backward directions. A different augmentation would introduce stages for reformulating problems when they prove difficult to solve (MacLellan, 2011; Riddle, 1990) and for translating solutions in the reformulated space back into the original one. This will require specifying reformulation operators that alter the representations for states, goals, or operators. An even more radical extension would provide the framework with an ability to generate new control rules and thus explore the space of strategies automatically. Each of these extensions would involve incremental additions to the architecture rather than a major revision, which further suggests the benefits of our modular framework.

7. Concluding Remarks

In this paper, we noted that, although the literature on problem solving in humans and machines describes many approaches, most implemented systems adopt a single, unvarying strategy. In response, we presented a theoretical framework that accounted for this variation as differences in strategic knowledge that is interpreted, along with domain knowledge, by a problem-solving architecture. The architecture decomposes each problem into an intention, a down subproblem, and a right subproblem, and it operates in five stages that focus, in turn, on problem selection, intention generation, subproblem creation, failure checking, and termination checking.

We demonstrated that an implementation of this framework solves problems in seven distinct domains and that it supports 12 major problem-solving strategies, along with many minor variations. We saw that some of these map directly onto familiar techniques that have appeared widely in the literature, with most others being minor variations on them. However, we also showed that one strategy – combining forward chaining with least commitment – was novel and interesting, showing that our framework has the power to predict new techniques. We also carried out experiments with the various strategies, finding that the new scheme fared surprising well compared to more standard methods and showing the framework's support for empirical studies.

We examined a number of earlier architectures that have supported flexible approaches to problem solving, noting that our framework is similar in spirit to these precursors but that it addresses some distinctive issues. Finally, we acknowledged that both our framework and the FPS system remain limited in their coverage, and we outlined some additions that should improve their explanatory range. Our architecture already reproduces a wide variety of search behaviors, but such extensions would extend its account of problem solving substantially.

Acknowledgements

This research was supported by Grant No. N00014-10-1-0487 from the Office of Naval Research. We thank Jaime Carbonell, Steve Minton, Peter Stone, Manuela Veloso, and Subbarao Kambhampati for providing information about earlier work in the same tradition, as well as Colin Walker and Trevor Gee for their efforts on earlier versions of the system.

References

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

Chakrabarti, P. P., Ghose, S., & Desarkar, S. C. (1986). Heuristic search through islands. *Artificial Intelligence*, *29*, 339–348.

Clocksin, W. F., & Mellish, C. S. (1981). *Programming in Prolog*. Berlin: Springer-Verlag.

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

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.

Hoffmann, J. (2001). FF: The Fast-Forward planning system. *AI Magazine*, *22*, 57–62.

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

Kambhampati, S., & Srivastava, B. (1995). Universal Classical Planner: An algorithm for unifying state-space and plan-space planning. *Proceedings of the Third European Workshop on Planning Systems*. Assisi, Italy.

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.

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

MacLellan, C. (2011). An elaboration account of insight. *Proceedings of the 2011 AAAI Fall Symposium on the Advances in Cognitive Systems*. Arlington, VA: AAAI Press.

Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, A. (2001). The SHOP planning system. *AI Magazine*, *22*, 91–94.

Newell, A., Shaw, J. C., & Simon, H. A. (1957). Empirical explorations of the Logic Theory Machine. A case study in heuristic. *Proceedings of the Western Joint Computer Conference* (pp. 218–230) New York: Institute of Radio Engineers.

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, Paris.

Newell, A., & Simon, H. A. (1972). *Human problem solving* Englewood Cliffs, NJ: Prentice-Hall.

Ohlsson, S. (1982). On the automated learning of problem solving rules. *Proceedings of the Sixth European Meeting on Cybernetics and Systems Research*.

Penberthy, S. J., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. Proceedings of the third international conference on knowledge representation and reasoning (pp. 103–114). Cambridge, MA: Morgan Kaufmann.

Riddle, P. J. (1990). Automating problem reformulation. In D. P. Benjamin (Ed.), *Change of representation and inductive bias*. Boston: Kluwer Academic Publishers.

van den Briel, M., Nigenda, R. S., Do, M. B., & Kambhampati, S. (2004). Effective approaches for partial satisfation (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. San Jose: AAAI Press.

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