Advances in Cognitive Systems 1 (2012) 1-18 Submitted 9/2012; published 12/2012

A Generative Theory of Problem Solving

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

Pat Langley LANGLEY@ASU.EDU
Collin Walker COLLIN.C.WALKER@ASU.EDU
Computer Science and Engineering, Arizona State University, Tempe, AZ 85287 USA

Abstract

Problem-solving strategies exhibited by humans are diverse and vary along a number of dimensions. In this paper, we attempt to address the problem of understanding and reproducing the great variety of problem-solving strategies observed in humans and existing artificial intelligence systems. To accomplish this task we propose a generative theory of problem solving, which defines the orthogonal dimensions over which problem-solving strategies can differ. Once the dimensions are defined, we suggest a number of possible values that each dimension can take, creating a space of strategies. This space covers problem-solving systems from multiple paradigms and provides a framework for combining these strategies in novel ways.

1. Introduction

The ability to solve complex problems is one of the hallmarks of human intelligence. Not only are humans capable of solving problems of overwhelming complexity, but they employ a great variety of strategies in doing so. Because the cognitive systems community is interested in the development of flexible and general problem-solving systems, a theory that accounts for this variety should be of great interest. Early work on problem solving focused on studying human think-aloud protocols (Newell & Simon, 1961) and resulted in the development of the General Problem Solver (GPS), as well as the introduction of means-ends analysis. More recent work has explored the space of plausible, but not necessarily human, strategies and has resulted in computational approaches to problem solving (e.g., forward chaining and systematic planners). A general trend in these research efforts has been movement towards the discovery and development of new strategies, but little effort has been directed toward general theories that organize them.

There are many possible paths to constructing such a theory, but the history of science suggests a reliable four-step approach:

- Collect phenomena you seek to explain
- Generate a theory that describes these phenomena

- Ensure the theory explains the phenomena you have collected (coverage)
- Use the theory to predict new phenomena that have not yet been observed

This general approach to constructing theories has resulted in numerous well-known theories, including Mendeleev's periodic table of elements (Mendeleev, 1879) and the Standard Model of particle physics (Oerter, 2006). In our case, the approach differs slightly from historical accounts; instead of collecting natural phenomena, we collect known strategies of problem solving employed by humans or artificial intelligence systems. Once we have collected these strategies, we induce a theory that explains them and outlines how they can be combined to produce new strategies.

The theory we propose constitutes a step toward a deeper theoretical understanding of the problem-solving strategies observed in humans and artificial intelligence (AI) systems. We will also make two explicit claims about the theory: that it has coverage of strategies from multiple paradigms present in the literature and that it provides a framework for combining these strategies in novel ways. Other important adjuncts to the theory are commitments to the use of a declarative formalism and to representing strategic knowledge as general, meta-level rules that include no domain predicates. By representing domain-specific and domain-general (also referred to as meta-level or strategic) knowledge declaratively it is possible to monitor and reason about this knowledge (Cox, 2011).

We have chosen to organize our presentation in a way that parallels the four steps described above. In Section 2, we describe the phenomena that we wish to explain, in this case known problem-solving strategies. In Section 3, we discuss a theory that explains these strategies, while in Section 4 we demonstrate the theory's coverage by showing the rules that produce the behavior of four problem-solving strategies and discuss the predictive abilities of our framework, showing the rules for a novel strategy. In Section 5, we discuss related work and compare it to our approach, while Section 6 outlines directions for possible future work and concludes the paper.

2. The Variety of Problem-Solving Strategies

The first step in constructing a theory is to collect the phenomena that we wish to explain. In this case the phenomena are the problem-solving strategies that have appeared in the literature. In this section, we review some strategies that have influenced the development of our modular theory. First, we will review Eureka (Jones & Langley, 1995), which finds its roots in human problem solving. Next, we outline the approach taken by Fast-Forward (FF) (Hoffmann, 2001), which is oriented toward efficient search. Finally, we describe the UCPOP algorithm (Penberthy & Weld, 1992), which combines elements of both and introduces some novel behavior.

The first problem solving strategy we will describe is the one employed by the Eureka system, which was designed to mimic the human ability to reason analogically. This system searches a state space given a set of operators and goals. When the system identifies that there are unsolved goals it queries its analogical memory, which maps states to operators, and stochastically chooses an operator to apply. If the operator is applicable, it is applied and the system is recursively called on the resulting state. However, if the operator is not applicable, then the system generates subgoals corresponding to the unsatisfied preconditions of the operator and then is recursively called on the current state and the generated subgoals. Once the subgoals are satisfied the system executes the

previously selected operator. If the system is unable to satisfy the subgoals, then the system starts search over from the initial state. Since Eureka does not require that selected operators achieve a goal, its search can operate bi-directionally. Also, the system commits to restarting search whenever a failure is encountered. This non-systematic approach (some search may be duplicated), known as iterative sampling (Langley, 1992), enables the system to flexibly explore the search space, not getting stuck in a portion of the search tree.

Another strategy is used by the Fast-Forward (FF) planning system (Hoffmann, 2001), which aims to perform efficient plan generation by carrying out forward search in the state space. The system starts with an initial state, chains over the operators that are applicable, and searches for a state where the goals are satisfied. The particular search technique used is called enforced hill climbing (Hoffmann, 2010), which greedily chooses states that are closer to the goal then the parent. Unlike conventional hill climbing, whenever there are no children that are closer to the goal breadth-first search is used to identify a child that is better. This prevents the search from getting stuck in a local maxima. Because the system forward chains, only operators that are applicable in the current state are chosen for application. This strategy has proven to be very effective in planning competitions and exemplifies much of the recent work on problem-solving systems.

A third strategy that we will examine is that employed by UCPOP (Penberthy & Weld, 1992), a system that incorporates elements of both human and artificial problem solving. This system, similar to the previous two systems, searches a state space given a set of operators and goals. Once these are specified, the system backward chains, non-deterministically choosing operators that satisfy goals. If a chosen operator is not applicable, then the system generates new goals corresponding to the unsatisfied preconditions of the operator and the process is repeated. During this process the system keeps track of "threats,"or negative interactions between selected operators, and generates ordering constraints that prevent negative interaction. The UCPOP system also employs the systematic A* search technique (Hart, Nillson, & Raphael, 1968) when making decisions about which operators to select. Unlike the previous strategies, UCPOP takes a least-commitment approach to operator ordering.[1]

To review, the Eureka system utilizes a non-systematic goal-driven problem solving strategy that mimics human analogical problem solving. The FF system, which is motivated by efficiency concerns, uses systematic forward search. Finally, the UCPOP system combines elements of both paradigms, incorporating a systematic, goal-driven scheme but also adopting a delayed commitment approach to operator ordering. These strategies are only a few of the many that appear in the literature, but we maintain that they cover a broad spectrum of behavior that would benefit from deeper theoretical understanding.

3. A Theory of Problem Solving

In the previous section, we reviewed three interesting problem-solving strategies, but these are just a few of the many known methods. Despite the great number of strategies in the literature, there are few theories of problem solving that account for this variety. We should briefly note the distinction between a theory and the models it produces. A theory provides an abstract framework that makes a

1. If this system were to interact with the physical world it would have to decide on an explicit ordering at runtime.

number of assumptions about how systems of a certain class behave. In contrast, a model introduces auxiliary assumptions that make the theory operational and let it make contact with observations. In many fields, a theory provides a set of parameters and specific models provide their values.

Thus, a theory of problem solving might take the form of parameters that characterize the space of methods, whereas particular models would commit to parameter values that produce specific strategies. In this section we propose such a generative theory of problem solving. We start by explaining the principles that underly our theoretical framework, then discuss an implemented system that embodies it and that supports the creation of models for different strategies.

3.1 Theoretical Principles

Before we state the contributions of our 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 (1958a), but it has become so commonly accepted that few now question whether it has potential for elaboration and extension.

The standard 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 assumptions, but it also moves beyond them to incorporate new postulates. These include:

- 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 intention or operator instance *I*; a right subproblem, which is a subproblem that has the same goals as *P*, but has a state that results from the application of *I* to *P*; a down subproblem, which is a subproblem that shares *P*'s state but has preconditions corresponding to *I*'s preconditions; and the solution to *P*'s subproblems. In the terminal case, a problem solution can also be a problem *P* that is marked as done (see Figure 1 for a graphical representation).
- Problems and their (attempted) solutions reside in a working memory that changes over the course of problem solving, whereas operators and strategic knowledge reside in a long-term memory that changes gradually if at all.
- The problem-solving process operates in cycles that involve five stages: problem selection, termination checking, intention generation, failure checking, and intention application. Each stage involves changes to the problem structures in working memory.
- Alternative problem-solving strategies result from variations on these five processing stages, with their settings being entirely independent of each other.

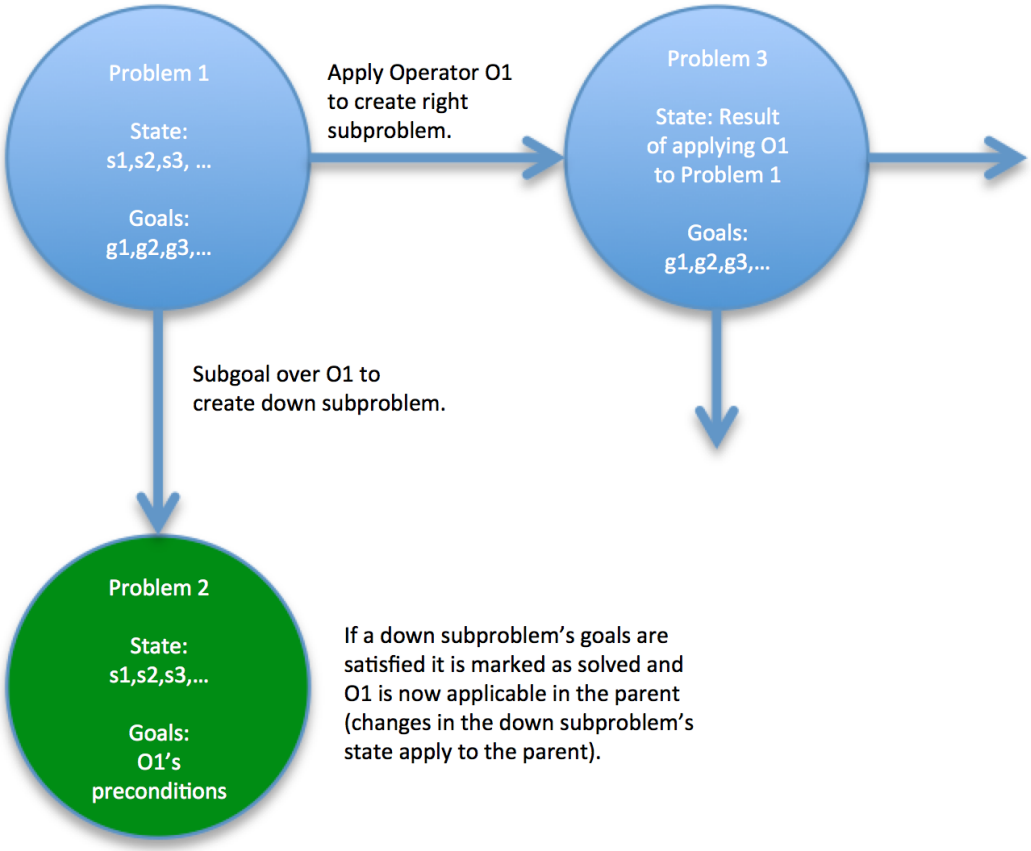

Figure 1. This is an example of a problem and its down and right subproblems. Once an intention to apply Operator 1 has been generated for Problem 1, a down subproblem is created that has goals that correspond to O1's preconditions. After this subproblem is solved (e.g., the goals are all satisfied) then the operator can be applied in the parent generating a right subproblem, which has Problem 1's goals and an updated state. A problem is also solved when its subproblems are solved. The process continues until Problem 1 is solved.

Although the first three assumptions specify important commitments about representation and organization, the final two tenets are the most interesting and important. The postulation of five stages that take on orthogonal settings provides the generative power to explain the great variety of possible problem-solving strategies. Thus, we should consider each stage and its possible settings in more detail.

Each cycle begins with the problem solver selecting a problem on which to focus its attention. If only one problem exists at the outset, then there is little choice, but this changes in later rounds. This stage examines working memory to check which problems are available and alters it to reflect the newly selected focus. Different settings for this step produce 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, whereas selecting the least recently created problems produces breadth-first search.

Once the problem solver has selected a problem, the second stage sees whether any work is needed. This compares the state and goal descriptions to determine the extent to which they match. If so, then it adds this information to working memory, which makes later stages on the cycle inoperative. Different settings for this step produce different criteria for deciding when a problem is solved, such as requiring that all goal elements are matched or, in cases where goals can be partially satisfied, that some threshold is exceeded.

If the problem solver determines that work remains on the current problem, the third stage selects an operator instance that it intends to apply to this end. We will refer to these as *intentions* because, at this point, they are only potential; the problem solver may not apply them on this cycle or at all. This stage examines both the current problem and the available operators, and it adds one intention to working memory that is associated with the problem, if there are any valid intentions. If no operator instances are available (e.g., if all candidates have failed on earlier attempts), the problem solver annotates the problem to reflect that. Different settings for this step produce alternatives like backward chaining from goals and forward chaining from the state.

The fourth stage checks for failures that indicate the problem solver should abandon its choices. This process involves inspecting the current problem and its associated intentions and then, if it detects issues, adding information to working memory that marks the problem as failed. For example, this stage may check for a lack of available intentions (an annotation from the previous step), loops in the problem states considered along the current path, subproblems that are more difficult than their parents, that the maximum search depth has been reached, or even that too many attempts have been made to solve a task. Such tests are not usually associated with problem-solving strategies, but they are just as important as more constructive aspects.

Finally, if the problem solver has not abandoned its current approach, it "applies" an intention to the current problem *P* in the fifth stage (see Figure 1). The application of an intention consists of subgoaling over the intention, generating a down subproblem, a problem with the same state as *P*, but with goals based on the intention's conditions. In the case where the intention is applicable, the down problem is trivially solved. More interesting are the conditions under which a right subproblem, or a problem which has the same goals as P but with a state modified by the effects of the applied intention, is generated. There are two different settings that generate two distinct types of behavior. For instance, an eager commitment setting causes the system to check to see if the preconditions of intentions are met in P's state description. If they are, then one applicable intention is chosen and the problem solver creates a right subproblem. However, if none of the intentions are applicable, then a right subproblem is not created, and the system waits to solve a down subproblem, which will make an intention applicable. Another setting, for delayed commitment, causes the system to prefer subgoaling and only applying intentions once there is no more subgoaling to perform and all down problems are solved.

In summary, the proposed theory consists of five stages: problem selection, which selects the problem to focus on; termination check, which determines if the selected problem is done; intention generation, which creations intentions to apply operators and adds them to the current problem; failure check, which ensures that a problem is marked as failed if it forms a loop, exceeds a depth

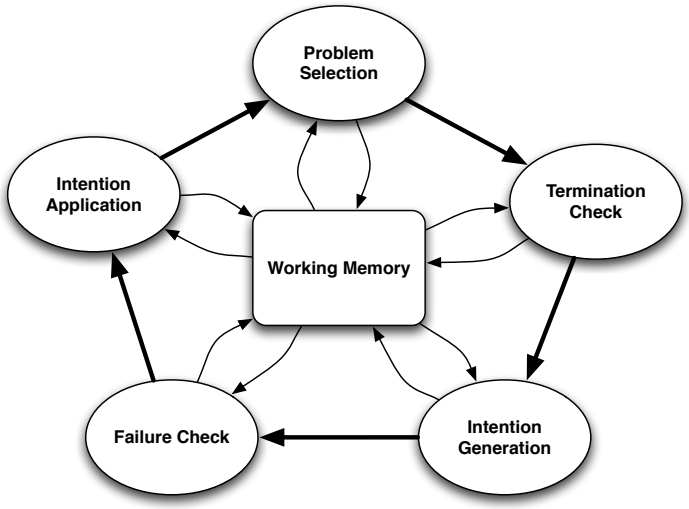

Figure 2. The GTPS outer control loop. Each cycle starts at the problem selection module. By specifying a setting for each module (e.g., depth-first problem selection, all goals satisfied termination check, means-ends intention generation, loop and dead-end failure check, and eager commitment intention application) GTPS reproduces the behavior of a particular strategy (for these settings it would reproduce the behavior of GPS).

limit, or is a dead-end; and intention application, which applies or subgoals intentions. These five stages are orthogonal, making the system generative in character and providing the power to produce a great variety of problem-solving behavior.

3.2 A Declarative Implementation of the Theory

Although the theoretical assumptions just discussed appear to offer an account for variations among problem-solving strategies, we cannot evaluate this ability in the abstract. For this, we must instantiate the general claims within some operational framework that can actually carry out these strategies.

To this end, we have developed a flexible software framework that we call GTPS, for Generative Theory of Problem Solving (shown in Figure 2). This system includes separate knowledge bases for each of the five stages (not shown in Figure 2), each of which is populated with declarative rules, written in Answer Set Prolog (Gelfond & Lifschitz, 1988), that formalize settings like those described earlier. GTPS also incorporates a high-level wrapper, written in C#, that inputs the initial problem state and goals, populates the working memory, and repeatedly cycles through the five stages until it solves the problem or decides it has failed. For each stage, the wrapper loads the associated rules and passes them, along with the elements in working memory, to an answer set solver. The answer set solver returns a number of predicates, some of which are adds and deletes, which specify predicates that should be added and deleted from working memory. The wrapper applies these adds and deletes to the working memory and ignores the remaining predicates.

Our use of Answer Set Prolog also supports two features that, although not core tenets of our theory, we believe are important adjuncts. The first is that the settings for each problem-solving stage are encoded in declarative form, which enables monitoring and reasoning about the settings[2] and also provides a fine-grained level of analysis that can model minor variations among strategies. Two other notable systems that have made this commitment are PRODIGY (Carbonell, Etzioni, Gil, Joseph, & Knoblock, 1991; Fink & Veloso, 1996) and Soar (Laird, Newell, & Rosenbloom, 1987).

The second is that Answer Set Prolog provides the representational power needed to state meta-level rules that specify problem-solving strategies in a very general way. Although GTPS makes use of domain predicates in representing states, goals, and operators, the rules that determine its behavior at each stage do not refer to them. This contrasts with the substantial body of work on search control that uses domain-specific knowledge to guide behavior. Examples include TLPlan (Bacchus & Kabanza, 2000) and SATPLAN (Kautz & Selman, 1998), and even papers on Soar and Prodigy have emphasized this aspect, despite their ability to encode meta-level rules.

We should note that answer set solvers (Gebser, Kaufmann, Neumann, & Schaub, 2007; Lierler & Maratea, 2004) have previously been used on multi-step planning problems, but these are typically called as a black box that inputs a task and returns a set of solutions. This approach does not provide the fine-grained control needed to explain variations across problem-solving strategies that is our aim. For this reason, we use answer set methods only to carry out inferences in support of each problem-solving stage. This both limits the complexity of the inference process and provides the granularity that we need.

4. Evaluation

The proposed theory, and the system that embodies it, are motivated by a desire to understand and reproduce the great variety of problem solving strategies used by both human and computational systems. For a system with these goals, there exists no standard criteria for evaluation. In this respect, we refer to history of science, which suggests two criteria that have been used to evaluate proposed theories in the past: coverage and predictive abilities. Thus, we strove to construct a theory that would do well on both dimensions. In this section, we evaluate our system against both of these criteria.

4.1 Coverage

The first, and primary claim, of our theoretical framework is that it has coverage of problem-solving strategies from multiple paradigms present in the literature. By coverage, we mean that the system is capable of reproducing the behavior of these strategies. This metric is both relevant and important because it shows how effectively our theoretical framework explains the phenomena in question, in this case problem-solving strategies. An authoritative evaluation would require a demonstration of the ability to reproduce the tens to hundreds of strategies present in the literature, but this would be a tremendous undertaking. As a first step in this direction, we have selected four problem-solving strategies from the literature that are representative of some of the major paradigms present in the

2. We do not currently have rules that determine which settings to use, but this is a direction for future work that will require a declarative representation.

literature. Demonstrating that our system can reproduce the behavior of these strategies is evidence that we could produce the behavior of other strategies within these paradigms.

Table 1. Table of crucial rules for the GPS/STRIPS model. This model also uses standard rule sets for termination checking (all goals satisfied) and failure detection (loops, dead-ends, and depth limit detected), but they are not shown to save space.

Module	Primary Rules		
problem selection	delete(selectedProblem(P))	:-	selectedProblem(P).
	add(selectedProblem(P))	:-	problem(P), not failedProblem(P), not done(P),
	<*empty head*>	:-	add(selectedProblem(P)), problem(Q), not failedProblem(Q), not done(Q), depth(P,D), depth(Q,R), not ancestorDone(Q), R > D.
intention generation	1{add(intendToApply(P,O))} 1	:-	reducesDifference(P,O,D), selectedProblem(P).
	add(allIntentionsGenerated(P))	:-	selectedProblem(P), not intentionsAvailable(P).
intention application	atLeastOneApplicableIntention(P)	:-	intendToApply(P,O), applicableIntention(P,O).

Our first model focuses on the strategy used by the GPS (Newell, Shaw, & Simon, 1958b) and STRIPS (Fikes & Nillson, 1972) systems, seminal works in psychology and AI. This strategy consists of a rule set for each of the five stages of our framework: depth-first problem selection; all goals satisfied termination checking; means-ends intention generation; loops, dead-end, and max depth failure detection; and eager operator application. We will take a closer look at each of the rules constituting this model, shown in Table 1. Demonstration of this strategy will show that our system can produce the behavior of backward chaining strategies.

The first module of GTPS is problem selection, which for this model involves a depth-first approach to selecting a problem. This behavior is captured by three rules: one that creates a delete to remove the currently selected problem from the working memory, a second that creates an add to select a new problem that is not failed or done, and a third to constrain the problems that can be added, to ensure there are no other valid (not failed or done) problems that are deeper then the problem selected (a solution is invalid if the body is true because the system will derive false, or

an empty head). There are also various helper rules, not shown, that ensure only one problem is selected at a time and that produce the helper predicates needed to fire these rules.

Once GTPS selects a problem, it uses the model's termination rules to detect if all the goals are satisfied in the selected problem and marks the problem as done, or solved, if this is the case – the remaining modules are skipped if the problem is marked as done. This is the standard functionality of most problem-solving strategies.

After GTPS determines that the currently selected problem is not done, it moves on to intention generation, which in this model involves means-ends analysis. We can see two crucial rules that produce this behavior: the first adds an intention that reduces a difference, or unsatisfied goal, in the problem (the 1's ensure that one and only one intention is added at each step– without them all intentions would be generated) and the second annotates the state if all of the possible intentions have been generated for that state.

After intention generation, GTPS executes the failure detection rules, which check for loops, dead ends, and problems that exceed the depth limit in the set of expanded problems. As with termination checking setting, which checked to see if all goals are satisfied, this is a standard approach in many problem-solving system, although such details are usually omitted from system descriptions.

Finally, the system applies intentions in the application module, which for this model takes an eager approach. The primary rule here produces the atLeastOneApplicableIntention predicate that causes the module to apply one of the applicable intentions. If this predicate is not produced then the module will subgoal over a non-applicable intention, if one exists. The combination of these rule sets and our theoretical framework produces a model that reproduces the basic behavior of GPS and STRIPS (i.e., backward-chaining). It identifies differences that must be reduced, applies operators to reduce them, subgoals where operators are non-applicable, and uses depth-first search to explore the space.

Next we will demonstrate how our framework can model a modified version of the Eureka system that we call Eureka/, which is modified to use means-ends analysis without heuristic information from the analogies. Coverage of this strategy will show that our system is capable of producing the behavior of non-systematic approaches to search. Table 2 shows the crucial rules for the Eureka/ model.

This table shows the rules for problem selection, the only module that differs from the GPS/STRIPS model. Even so, the careful reader will notice that the rules are almost identical to those for problem selection in the earlier model. This is because iterative sampling is very similar to depth-first search, with only the second and third rules being modified to prevent selection of problems (excluding the root) that have been part of a failed solution attempt. This causes the search to restart after each failure instead of backtracking. The end result of these changes is a model that has the behavior of Eureka/; the system selects operators that achieve unsatisfied goals, applies them, subgoaling if they are non-applicable, and restarts search if a failure is encountered.

Next, we will show how our theory can vary along the dimension of intention application by modeling the UCPOP/ strategy, a version of UCPOP modified to use depth-first search instead of A*. Coverage of this strategy will show that our system can reproduce the behavior of strategies in the partial-order planning paradigm. The critical rules for the UCPOP/ model appear in Table 3. The

Table 2. Table of crucial rules for the Eureka′ model. All rules are the same as those used in the GPS/STRIPS model, except for problem selection.

Module	Primary Rules		
problem selection	delete(selectedProblem(P))	:-	selectedProblem(P).
	add(selectedProblem(P))	:-	problem(P), not failedProblem(P), not done(P), not inFailedLine(P).
	<*empty head*>	:-	add(selectedProblem(P)), problem(Q), not failedProblem(Q), not done(Q), not inFailedLine(Q) depth(P,D), depth(Q,R), not ancestorDone(Q), R > D.

Table 3. Table of crucial rules for the UCPOP′ model. All rules are the same as those used in the GPS/STRIPS model, except for the intention application.

Module	Primary Rules		
intention application	atLeastOneApplicableIntention(P)	:-	intendToApply(P,O), applicableIntention(P,O), allDifferencesCovered(P).
	atLeastOneApplicableIntention(P)	:-	intendToApply(P,O), applicableIntention(P,O), allIntentionsGenerated(P).

Table 4. Table of crucial rules for the FF/ model. All the rules for this model are the same as those in the GPS/STRIPS model, except for its forward chaining approach to intention generation.

Module	Primary Rules		
intention generation	1{add(intendToApply(P,O))} 1	:-	operatorApplicable(P,O), selectedProblem(P).
	add(allIntentionsGenerated(P))	:-	selectedProblem(P), not intentionsAvailable(P).
intention application	atLeastOneApplicableIntention(P)	:-	intendToApply(P,O), applicableIntention(P,O).

new model is similar to those we have already considered. Both GPS/STRIPS and UCPOP/ take a depth-first approach to problem selection and therefore share the rules responsible. Furthermore, all three models adopt means-ends analysis and thus hold those rule sets are in common. The primary difference lies in UCPOP/'s approach to intention application, which relies on delayed commitment. This involves altering the rules for eager commitment so that they produce the atLeastOneApplicableIntention predicate only when all differences have been covered or all possible intentions have been generated. This change causes the system to delay application of any intentions until no more subgoaling or intention generation can be performed, resulting in a model that reproduces the behavior of the UCPOP/ system; it selects operators to achieve unsatisfied goals, prefers to subgoal until all goals are supported, and uses systematic search.

Finally, we will examine how our framework produces the behavior of FF/, a modified version of the FF strategy that uses depth-first search instead of enforced hill climbing and that does not use any heuristic for intention generation. Table 4 shows the crucial rules for this model. Demonstration of this strategy shows that our system can reproduce the behavior of forward chaining systems. Like GPS and UCPOP/, this model utilizes depth-first search, and like GPS and Eureka/, it uses eager commitment to apply intentions. However unlike the previous models, this strategy uses forward chaining instead of backward chaining to generate intentions. These settings result in a model that searches systematically for solutions by forward chaining and applying operators eagerly, just as FF/ does.

We have demonstrated how our framework supports the GPS, Eureka/, UCPOP/, and FF/ models, showing that our system can reproduce the behavior of forward and backward chaining systems, systematic and non-systematic systems, and eager and delayed commitment systems. These six paradigms encompass a large body of strategies that we believe our system should be capable of reproducing. However, modeling them precisely would entail formalizing new rule sets for problem selection (e.g., A* or enforced hill climbing) and intention generation (e.g., analogical selection).

4.2 Predictive Abilities

The second claim about our theory is that it provides a framework for combining strategies from different paradigms in novel ways. In other words, in addition to covering many strategies from the

literature, the theory also generates novel problem-solving behavior. This is an important metric for evaluation of our system because if we can predict reasonable strategies, this suggests that the framework has captured the order underlying the phenomena, which is an objective of any theory.[3] To evaluate these predictive abilities, we will outline the rule sets that we have constructed for each module of our framework and use these sets to demonstrate the combinatorial nature of our generative theory – in a similar way to generative grammars in linguistics. In addition to showing how the strategies we have reviewed fit into the generated theories, we will show the novel strategies that our theory suggests, which are hybrids of the discussed strategies, and discuss the behavior of one of these strategies.

We will begin by reviewing the rule sets we have formalized for each of the modules of our system, which provide the foundation for the generative abilities of our theoretical framework. To give an overview, we have implemented three approaches for problem selection, one approach for termination checking, three approaches for intention generation, one approach for failure checking, and two approaches for intention application. All but two of these rule sets has been demonstrated in section 4.1. The first is a new rule set for problem selection, which randomly selects valid problems from the space to focus on. The second is a new rule set for intention generation, which randomly generates valid intentions, that have not already been tried and failed. We initially constructed these rule sets for testing purposes, but we believe that they produce interesting behavior.

By combining the set of rules that we have formalized and our theoretical framework, we have defined a space of models where each point in that space is a commitment to a particular rule set for each of the modules.[4] In section 4.1, we demonstrated that this space covers a diverse set of strategies from multiple paradigms present in the literature, also interesting are the models that do not correspond to these strategies. To explore these further, we have enumerated the list of problem-solving strategies our system is capable of modeling in Table 5. In addition to simply listing the models, we have also annotated them with the reviewed strategies they correspond to; this demonstrates that in addition to reproducing the behavior of the given strategies (GPS, Eureka/, UCPOP/, and FF/) our system also generates 14 strategies that are variations and combinations of these strategies. This demonstrates that our theory has the ability to combine strategies from different paradigms to produce novel behavior.

One model produced by GTPS uses iterative sampling, like Eureka/; means-ends, like GPS/STRIPS and Eureka/; and delayed operator application, like UCPOP/. The resulting behavior of this new strategy selects operators that achieve unsatisfied goals, restarts on failure, and only applies operators when subgoaling is not possible. This strategy seems trivial because it extends the previous strategies by combining aspects of each, but it shows that our system is capable of unifying the strengths of many strategies to generate new behavior, showing that the theory has captured the underlying order of the phenomena.

3. One key difficulty with this metric is that it is not clear what constitutes a "reasonable," "meaningful," or "interesting" strategy. Following the approach taken in evaluating theories of human grammar, we decided that a valid strategy was one that seems reasonable to an AI researcher.
4. Similar to generative grammars in linguistics, our system is combinatorial in character and the number of models our theory can produce will grow quickly as more rule sets are constructed for each of the modules.

Table 5. This table shows the models the system is capable of generating. Appended to the table are three additional rows outlining the settings for the Eureka, UCPOP, and FF systems (A*, analogical selection, and enforced hill climbing have not been implemented.). Note, the columns for the termination and failure checking are omitted to save space since only one approach has been formalized for each.

Strategy Name	Problem Selection	Intention Generation	Intention Application
FF/	depth-first	forward chaining	eager
	depth-first	forward chaining	delayed
GPS	depth-first	means-ends	eager
UCPOP/	depth-first	means-ends	delayed
	depth-first	random valid intention	eager
	depth-first	random valid intention	delayed
	iterative-sampling	forward chaining	eager
	iterative-sampling	forward chaining	delayed
Eureka/	iterative-sampling	means-ends	eager
	iterative-sampling	means-ends	delayed
	iterative-sampling	random valid intention	eager
	iterative-sampling	random valid intention	delayed
	random valid problem	forward chaining	eager
	random valid problem	forward chaining	delayed
	random valid problem	means-ends	eager
	random valid problem	means-ends	delayed
	random valid problem	random valid intention	eager
	random valid problem	random valid intention	delayed
UCPOP	*A**	means-ends	delayed
Eureka	iterative sampling	means-end with *analogical heuristic*	eager
FF	*enforced hill climbing*	forward chaining	eager

4.3 Discussion

In this section, we have discussed the two explicit claims of our paper: that our theoretical framework has coverage of problem-solving strategies from multiple paradigms present in the literature and that it provides a framework to combine these strategies in novel ways. We demonstrated the coverage of our system by showing how it can reproduce the behavior of GPS, Eureka/, UCPOP/, and FF/, showing that our system can reproduce the behavior of forward and backward chaining systems, systematic and non-systematic systems, and eager and delayed commitment systems. Finally, we showed the generative abilities of our system by elaborating all of the models that our system is capable of producing and by looking at one novel strategy in further detail.

5. Related Work

Although computational accounts for variations in problem-solving strategies have been uncommon in the literature, our work is not the first on this topic. For instance, Kambhampati and Srivastava's

(1995) work on the Universal Classical Planning Algorithm has similar objectives. This attempts to unify "plan-space" and "state-space" approaches to planning by casting each as strategies for refining a partial order plan. Thus, it offers a framework that modeled a number of distinct strategies in terms of whether their refinement method involves forward search through a state space, backward search through such a space, or search through a plan space. Although similar in spirit, we maintain that our framework's multiple dimensions provide much wider coverage of the space of problem-solving strategies.

A second system that is similar to ours is Polyscheme (Cassimatis, 2002, 2005), a cognitive architecture that integrates multiple representation and inference schemes. This system defines two principles, one of which unifies many high-level cognitive modeling paradigms: the common function principle, which includes forward inference, subgoaling, simulating alternate worlds, and identity matching. This work is similar to ours in that it attempts to unify many problem-solving paradigms, but we propose multiple dimensions, which enables our system to be more generative and gives us broader coverage.

Another effort that supports a broad range of problem-solving behaviors revolves around the PRODIGY architecture (Carbonell et al., 1991; Fink & Veloso, 1996). This framework is more similar to ours in that it specifies alternative strategies in terms of declarative control rules. PRODIGY uses this knowledge at different stages of a decision-making loop to select among goals, states, operators, and bindings. These do not map precisely onto our five stages, but they play a very similar role. One important difference is that published work on this framework focuses on the use and acquisition of domain-specific control knowledge, whereas our framework emphasizes domain-independent strategies.[5]

The FLECS system (Veloso & Stone, 1995), which extends the PRODIGY framework, comes even closer to our approach by supporting both eager and delayed commitment strategies, an idea reflected in our application module. FLECS also provides the ability to shift between progression and regression approaches to planning, which maps directly on to settings for our framework's intention generation module. However, our theory also supports control over traversal of the problem space, checks for problem termination, and application of constraints. Thus, we can view it as an extended, more flexible version of FLECS that covers a wider range of problem-solving behaviors.

Another theoretical framework with similar aims is Soar (Laird et al., 1987), a cognitive architecture that organizes behavior around problem-space search. Like PRODIGY, it encodes strategic knowledge in terms of control rules, although these tend to occur at a finer level of granularity. Early research on Soar (Laird & Newell, 1983; Laird, 1984) examined the use of this approach to mimic a variety of "weak methods" that embody domain-independent problem-solving strategies, in much the same spirit as our own work, but more recent papers have emphasized the use and acquisition of domain-specific control knowledge. Our framework also supports dimensions of variation not studied in Soar, such as eager and delayed commitment, but the two theories still have much in common.

In summary, previous work on flexible problem solving has studied some of the issues addressed in this paper, but we believe that our theory offers advances over earlier approaches. In particular,

5. Minton (2012) reports that PRODIGY could utilize domain-independent control rules, but the project did not explore this ability systematically.

it combines an expanded set of dimensions that cover a wider space of problem-solving strategies with high-level, declarative rules for specifying these alternative cognitive behaviors.

6. Concluding Remarks

Humans are flexible problem solvers who can employ a great variety of problem-solving strategies. In this paper, we attempted to provide a deeper understanding of this variety by proposing a theory that specified five orthogonal facets of this cognitive activity: problem selection, termination checking, intention generation, failure checking, and intention application. Additionally, we described the theory's implementation in the GTPS framework, which uses declarative, meta-level rules to formalize the different settings for these dimensions. The generative, combinatorial character of our theory lets it account for a wide range of problem-solving behaviors.

In writing this paper, we made two explicit claims about the framework. The first was that it provides coverage of problem-solving strategies from multiple paradigms present in the literature, while the second was that it provides a framework for combining these strategies in novel ways. We also made a commitment to stating models of particular problem-solving methods in terms of declarative knowledge, which gives our framework the ability to monitor an reason about its strategic knowledge, something we plan on doing in future work.

We supported the first claim by showing that the framework was able to reproduce the behavior of four strategies. In particular, we reported the rules of four models – GPS, Eureka/, UCPOP/, and FF/ – that embody diverse approaches to problem solving. These four models demonstrate that our system can reproduce the behavior of forward and backward chaining systems, systematic and non-systematic systems, and eager and delayed commitment systems. We did not reproduce the behavior of all the strategies from these paradigms, but they seem within our framework and suggest an obvious direction for future work.

We supported our second claim about the generation of novel strategies by examining the set of models that GTPS can produce (Table 5). In addition to the four models just discussed, GTPS produces 14 additional models. One promising idea for future research is to carry out automated exploration of the models generated by the theory. This will be particularly useful as we add more settings for the various stages, and thus increase the number of models supported.

In conclusion, humans and artificial intelligence systems utilize a great variety of problem-solving strategies, but our theoretical understanding of these phenomena is limited. We have attempted to fill this gap by introducing a theoretical framework that specifies a space of strategies and that can reproduce the behavior of any strategy in the space. The theory covers a number of problem-solving strategies from multiple paradigms described in the literature, and it also provides a framework for combining these strategies in novel ways. In summary, we believe that our theoretical framework offers a path toward flexible and domain-independent problem-solving systems, which in turn should bring us closer to developing artifacts that exhibit human-level intelligence.

Acknowledgements

This research was supported by Grant No. N00014-10-1-0487 from the Office of Naval Research, by Grant No. CMMI-1002910 from the National Science Foundation, and by a gift from Lockheed

Martin Advanced Technology Laboratories. We thank Jaime Carbonell, Steve Minton, Peter Stone, Manuela Veloso, and Subbarao Kambhampati for providing information about the PRODIGY, FLECS, and Universal Classical Planner systems.

References

Bacchus, F., & Kabanza, F. (2000). Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, *116*, 2000.

Carbonell, J. G., Etzioni, O., Gil, Y., Joseph, R., & Knoblock, C. (1991). PRODIGY: An Integrated Architecture for Planning and Learning . *ACM SIGART Bulletin*, *2*, 51–55.

Cassimatis, N. L. (2002). *Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes*. Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Cassimatis, N. L. (2005). Integrating cognitive models based on different computational methods. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*.

Cox, M. T. (2011). Metareasoning, Monitoring, and Self-Explanation. In M. T. Cox & A. Raja (Eds.), *Metareasoning: Thinking about thinking* (pp. 131–149). Cambridge, MA: MIT Press.

Fikes, R. E., & Nillson, N. J. (1972). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, *2*, 189–208.

Fink, E., & Veloso, M. M. (1996). Formalizing the prodigy planning algorithm. *New Directions in AI Planning*, 261–271.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). clasp: A conflict-driven answer set solver. *Logic Programming and Nonmonotonic Reasoning*, 260–265.

Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. *Proceedings of the Fifth International Conference on Logic Programming*, *161*, 1070–1080.

Hart, P., Nillson, N. J., & Raphael, B. (1968, July). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on System Science and Cybernetics*, *4*(2), 100–107.

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

Hoffmann, J. (2010). A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. *Foundations of Intelligent Systems*, 691–701.

Jones, R. M., & Langley, P. (1995). Retrieval and Learning in Analogical Problem Solving. *Proceedings of the Seventeenth Conference of the Cognitive Science Society*, 466–471.

Kambhampati, S., & Srivastava, B. (1995, April). Universal Classical Planner: An algorithm for unifying State-space and Plan-space planning. *Current Trends in AI Planning*, 1–15.

Kautz, H., & Selman, B. (1998). The Role of Domain-Specific Knowledge in the Planning as Satisfiability Framework. In *Proceedings of the international conference on artificial intelligence planning* (pp. 181–189).

Laird, J. E. (1984, January). Universal subgoaling. *Carnegie Mellon University (thesis)*.

Laird, J. E., & Newell, A. (1983). *A universal weak method* (Tech. Rep. No. 1606). Carnegie Mellon University.

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. *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, 145–152.

Lierler, Y., & Maratea, M. (2004). Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs. In *Logic programming and nonmonotonic reasoning* (pp. 346–350). Berlin, Heidelberg: Springer Berlin Heidelberg.

Mendeleev, D. I. (1879). The Periodic Law of the Chemical Elements. *The Chemical News and Journal of Physical Sciences*, *40*, 244.

Minton, S. (2012). *Personal Communication.*

Newell, A., Shaw, J. C., & Simon, H. A. (1958a). Elements of a theory of human problem solving. *Psychological Review*, *65*(3), 151–166.

Newell, A., Shaw, J. C., & Simon, H. A. (1958b, December). Report on a General Problem-solving Program. *Proceedings of the International Conference on Information Processing*, 256–264.

Newell, A., & Simon, H. A. (1961). GPS, a program that simulates human thought. *Lernende automaten*, 109–124.

Oerter, R. (2006). *The Theory of Almost Everything: The Standard Model, the Unsung Triumph of Modern Physics*. Penguin Group.

Penberthy, S. J., & Weld, D. S. (1992). UCPOP: A Sound, Complete, Partial Order Planner for ADL. In B. Nebel, C. Rich, & W. Swartout (Eds.), *Proceedings of the third international conference on knowledge representation and reasoning* (pp. 103–114). San Mateo, California: Morgan Kaufmann.

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