Learning Expertise with Bounded Rationality and Self-awareness

Susan L. Epstein[1,2] and Smiljana Petrovic[2]

[1]Computer Science Department, Hunter College of The City University of New York
[2]Computer Science Department, The Graduate Center of The City University of New York
695 Park Avenue, New York, NY 10065, USA
susan.epstein@hunter.cuny.edu, spetrovic@gc.cuny.edu

Abstract

To address computationally challenging problems, ingenious researchers often develop a broad variety of heuristics with which to reason and learn. The integration of such good ideas into a robust, flexible environment presents a variety of difficulties, however. This paper describes how metareasoning that relies upon expertise, bounded rationality, and self-awareness supports a self-adaptive architecture for learning and problem solving. The resultant programs develop considerable skill on problems in three very different domains. They also provide insight into the strengths and pitfalls of metareasoning.

Anthropologists tell us that an *expert* is one who performs a task better and faster than the rest of us (D'Andrade, 1991). A programmed expert for challenging problems, however, is unlikely to be given every detail of its reasoning process in advance — rather, it is expected to learn its expertise on its own, to be *self-adaptive*. Ideally, expertise develops quickly. To accelerate its performance during both learning and testing, a self-adaptive system is likely to be subjected to *bounded rationality*, that is, to have limits placed on its space and time resources. As a result, computer scientists often construct *self-aware* programs that observe their own behavior and monitor their own reasoning to improve their performance, as in Figure 1 (Cox and Raja, 2007). The perils of such metareasoning become quickly evident in any ambitious application, however.

We believe that easy problems should be solved quickly, and that hard problems should take a bit longer. Rather than rely on thousands of learning experiences, the learners we describe develop considerable expertise after experience with relatively few problems. This paper recounts the challenges posed to one learning and problem-solving architecture by three different problem domains, and how metareasoning addresses those challenges successfully. The first section describes the architecture and the domains. The second section describes the premises that led to the architecture's structure. Subsequent sections explore the impact of bounded rationality, how to assess expertise, how to manage large bodies of heuristics to learn expertise, and how to think less but still maintain performance.

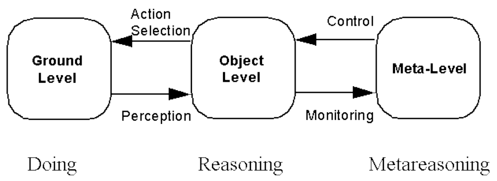

Figure 1: FORR addresses a problem. (Cox and Raja, 2007).

The Architecture and the Problems

FORR (For the Right Reasons) is a learning and problem solving architecture that models the development of expertise with metareasoning (Epstein, 1994a). From its experience on a set of problems, FORR learns to solve other, similar problems. Together, the problems it solves and those expected to be similar to them constitute a *problem class*. On any given problem, FORR seeks a sequence of actions that solves the problem, and can explain the reasoning that underlies its decisions.

FORR provides a flexible environment within which to design and execute experiments in metareasoning. The domain-dependent ground level in Figure 1 describes each world state as it appears during search for a solution. The object level re-represents and reasons about these perceptions, and gathers extensive data from its experience about the nature of the current problem class. The meta-level judges and possibly reformulates that decision-making process based upon what it detects within the object level.

FORR serves as a system shell. To produce a FORR-based application, one defines a *problem domain* (classes of problems that arise within it) and a set of heuristics that advise on decision making there. We focus here on three very different problem domains, each of which can be solved with a sequence of actions: game playing, path finding, and the solution of constraint satisfaction problems. Each domain has its own definitions for problem, problem class, action, and solution.

Hoyle is a FORR-based application that learns to play 19 two-person, perfect-information, finite-board games as well or better than the best human experts (Epstein, 2001). For Hoyle, a *problem class* is a game (e.g., tic-tac-toe), and a *problem* is a single playing experience. Each *action* is a legal move in the game. A *solution* is a sequence of such moves that achieves an *ideal outcome* (a win or a tie, as defined by the game tree). Given the rules of a game, Hoyle

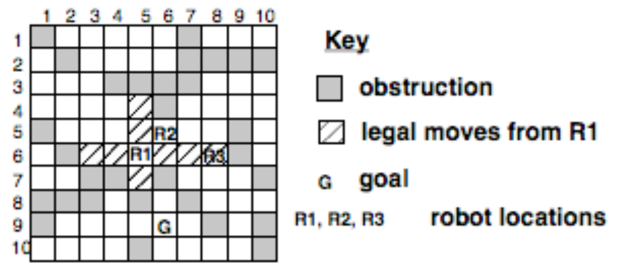

Figure 2: An example of Ariadne's grid world, where a robot *R* must move to the goal *G*. Problems used here are considerably larger (Epstein, 1998).

plays it against itself or against an external opponent and learns to play the game better. The games in Hoyle's repertoire are not as challenging as chess, but some have large game trees (as many as several billion nodes) and the expert programs crafted to serve as Hoyle's opponents required considerable game-specific knowledge and skill.

Ariadne is a FORR-based application that learns to find its way about a two-dimensional grid-based maze (Epstein, 1998). For Ariadne, a *problem class* is a maze generated with a specified size and difficulty; a *problem* is a trip in a given maze from an initial location to a goal, as in Figure 2. (The initial location and the goal vary from one problem to the next.) The learner has no map, only the coordinates of the goal and the current distance of the robot from the nearest obstruction in four orthogonal directions, as if the robot only "sees" certain positions on the grid. Each *action* moves the robot in a straight line to any such currently visible location. A *solution* is a sequence of such actions that reaches the goal. Given its lack of information, a robot can cycle repeatedly through the same locations. Represented as a graph, a moderately challenging 20 × 20 maze would have 280 nodes and about 1485 edges.

ACE (the Adaptive Constraint Engine) is a FORR-based application that learns to solve constraint satisfaction problems (*CSPs*) (Epstein, Freuder and Wallace, 2005). For ACE, a *problem* is an individual (binary) CSP: a set of variables, each with an associated set of values, and a set of constraints that restrict how pairs of variables can be bound to values simultaneously. A *problem class* is a set of CSPs with the same descriptive characterization (e.g., number of variables and maximum domain size).

Structured CSPs, like those in Figure 3, can have pockets of densely connected variables with very tight constraints. A metric that scores possible actions suggests a

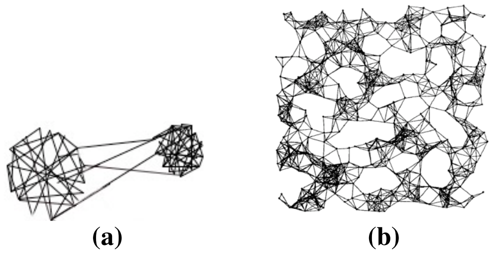

Figure 3: Examples of constraint graphs for (a) composed and (b) geometric problems (Johnson et al., 1989a). Each variable is represented as a node, and each constraint as an edge.

dual pair of heuristics: one takes the maximum, and the other takes the minimum. Structured problems present particular challenges for traditional solvers because the duals of traditional heuristics often fare better there than the original heuristics. Structured problems also are more similar to real-world problems than simple random CSPs. From extensive data on over 100 CSP classes, we report here on a composed class (Aardal et al., 2003) with 30 variables and domain size 6 (6^{30} nodes) and a geometric class (Johnson et al., 1989b) with 50 variables and domain size 10 (10^{50} nodes). Problems from both classes often stumped the best solvers at a recent CP competition.

Each *action* in ACE either selects a variable or assigns a value to a variable from its associated value set. After each assignment, propagation infers its effect on the domains of the as-yet-unassigned variables. If any variable is shown to have no values consistent with the current assignments (a *wipeout*), the most recent assignments are retracted chronologically until every variable has some possible value. A *solution* in ACE is a sequence of assignments that satisfies all the constraints and assigns every variable a value. Such search for a solution is NP-hard. CSPs model many of the most difficult problems solved by AI systems, including graph coloring, propositional satisfiability, and scheduling. Metareasoning supports learning to search them expertly.

Foundation Assumptions

FORR is based on a set of premises that dictate its general structure and behavior. The first set of premises addresses the reasons that underlie decisions during search:

• **Good reasons underlie intelligent actions.** In FORR, proposed domain-specific good reasons are called *Advisors.* Input to an Advisor is the current problem state and the available legal actions. Output from an Advisor is *advice* about some number of actions. For example, *Hurry* advises Ariadne to take long steps in any direction, and *Material* advises Hoyle to capture an opponent's piece.

• **A problem domain has many potential good reasons.** Typically, a host of Advisors can be extracted from human experts. Hoyle and Ariadne each have several dozen; ACE has more than 100 drawn from the constraint literature.

• **Some good reasons are always correct.** In FORR such reasons are *tier-1* Advisors that take priority during decision making. Their fast, accurate guidance chooses an action or eliminates some actions from further consideration. For example, it is always correct to make an immediately winning move in a game, to move directly to a visible goal in a maze, and to temporarily ignore a variable without neighbors in the constraint graph.

• **Some good decisions include more than one action.** In FORR this creates a set of *tier-2* Advisors that identify and address a subproblem with a *plan*, a (possibly ordered) set of actions. For example, Ariadne's *Roundabout* circumnavigates an obstruction that lies directly between the robot and the goal. Only one plan may be active at any point in time. The tier-1 Advisor *Enforcer* supports the execution of any active plan or terminates it for poor performance.

• **Most good reasons are fallible heuristics.** In FORR

```
***Address**(problem, class)*
    *state* ← start-state(*problem*)
    Until *problem* is solved or abandoned
    decision←consult-tier-1(*state, problem, class*)
    unless decision
        decision ←consult-tier-2(*state, problem, class*)
        unless *decision*
            decision←consult-tier-3(*state, problem, class*)
    state ← apply (*decision, state*)
```

Figure 4: **During search,** FORR addresses three tiers of Advisors in turn until a decision is made.

such reasons are *tier-3* Advisors that express preferences for choices numerically, as *strengths*. For example, Ariadne's *Hurry* assigns greater strengths to longer steps.

The *Address* pseudocode in Figure 4 coordinates all three tiers to make a decision; it is the object level in Figure 1. The next set of premises addresses how an appropriate combination of reasons is assembled:

• **Good reasons make different contributions on different problem classes.** For example, Ariadne's warehouse mazes have many objects to travel around, but its living-room mazes have few internal obstructions. FORR evaluates each tier-3 Advisor on each new problem class it encounters in a domain. Those that are more constructive we call *class-appropriate*; the others are *class-inappropriate.* For example, ACE's heuristics are provided in dual pairs and the program is left to sort out whether either is appropriate. Lines 1 and 2 in Table 1 show that duals are not necessarily consistent on different problem classes.

• **A combination of good reasons offers substantial benefits.** There is evidence for this both in people's reliance on multiple heuristics (Biswas et al., 1995; Crowley and Siegler, 1993; Ratterman and Epstein, 1995; Schraagen, 1993) and in programs that integrate multiple rationales to their advantage (Keim et al., 1999). Table 1 shows how pairs (lines 3-5) of heuristics, one for variable selection and the other for value selection, may outperform individual heuristics (lines 1-2).

• **Good combinations of reasons vary with the problem class.** Table 1 (lines 3-5) shows that pairs of Advisors successful on one problem class may do less well on another.

• **A program can *learn* to become an expert.** In a *run*, FORR addresses a sequence of problems from a class (the *learning phase*) and then evaluates its performance on a second sequence of problems (the *testing phase*) with learning turned off. All of Figure 1 is active during learning, but no further directives from the metareasoning module occur during testing. Because decisions may be nondeterministic, performance is evaluated over a set of runs (an *experiment*), as in Figure 5. The expectation is that relatively few learning problems will suffice. For example, ACE can learn to solve the geometric problems and the composed problems well after experience with only 30 of them. Table 1 shows how a mixture of heuristics learned by ACE (line 6) can outperform both individual heuristics and pairs of them.

```
***Experiment**(domain, class, number-of-runs)*
For number-of-runs
    Until learning is terminated              ;*learning phase*
        Select *problem* from *class*
        *Address(problem, class)*
        Learn from the solution, if any       ;*metareasoning*
    Until testing is terminated               ;*testing phase*
        Select *problem* from *class*
        *Address(problem, class)*
Average performance across all runs
```

Figure 5: Pseudocode for a FORR-based experiment.

Table 1: The appropriateness of a heuristic varies with the problem class, and multiple heuristics outperform individual ones in CSP search. Variable ordering with *ddd* calculates the ratio of the number of values that a variable could be assigned to the number of unassigned neighbors it has in the constraint graph. v_1 *and* v_2 are value-ordering heuristics. After propagation, v_1 orders values by the smallest domain size produced, and v_2 orders values by the largest product of domain sizes. Space is nodes searched; elapsed time is in CPU seconds.

	Geometric			***Composed***		
Heuristics	***Space***	***Time***	***Solved***	***Space***	***Time***	***Solved***
Min *ddd*	258.1	3.1	98%	996.7	2.0	82%
Max *ddd*	4722.7	32.3	6%	529.9	1.0	90%
Min *ddd*+v_1	199.7	3.6	98%	924.2	3.0	84%
Min *ddd*+v_2	171.6	3.3	98%	431.1	1.5	92%
Max *ddd*+v_2	3826.8	53.7	30%	430.6	1.4	92%
ACE mixture	146.8	5.1	100%	31.4	0.6	100%

We add one more premise, on representation:

• **Multiple representations enhance reasonin**g. Each Advisor in FORR has access to any representation of events and states from the ground level and the object level. These representations are shared and computed at the object level only on demand. For example, Hoyle represents a game board linearly, as a grid, as a set of patterns, and as a set of zones; Ariadne describes a maze in terms of gates, bases, barriers, and several kinds of corridors; and ACE has dozens of descriptions for a CSP, including many that identify kinds of subproblems and relationships among variables,

Other systems that rely on a mixture of heuristics combine them in a variety of ways. More complex heuristics may be reserved for the harder problems (Borrett, Tsang and Walsh, 1996). Heuristics from a *portfolio* may be selected to compete in parallel until one solves the problem, or used in turn on the same problem (Gagliolo and Schmidhuber, 2007; Gomes and Selman, 2001; Streeter, Golovin and Smith, 2007). A system may also label its heuristics individually for their appropriateness with learned *weights*. Heuristics can then be consulted one at a time in order of their weight (Minton et al., 1995; Nareyek, 2003) or they can *vote*, combining advice from each of them on every decision (Fukunaga, 2002). When it consults tier 3, FORR has its Advisors vote.

Learning with Bounded Rationality

Even under the control of parameters, a random problem generator cannot be trusted to create uniformly difficult problems in the same class. For example, the difficulty a fixed algorithm experiences within a class of putatively similar, randomly-generated CSPs may decrease not exponentially but according to a power law, that is, be *heavy-tailed* (Gomes et al., 2000). As a result, there are many much more difficult problems within the same class.

A reasonable response is to set some arbitrary standard for performance, and then insist upon it. One might, for example, allocate bounded resources (time or space) to a solver, and learn only from solutions achieved within them. The presumption here is that a solution that is arrived at quickly or visits fewer states is better. Setting such bounds is problematic, however. If they are set too low for typical problems in the class, only the easiest among them will be solved. Solutions will be fewer and training instances untrustworthy, because they have been drawn from problems where most any decision would do. If the resource bounds are set too high, however, long but successful searches will provide traces of not-so-expert behavior. For example, ACE can be given a limit on the number of *nodes* (partial assignments) it may search before it abandons a problem and moves on to the next one. Figure 6 shows the impact of a resource limit while learning to solve the geometric problems. A run was considered successful if solved at least 80% of its 50 testing problems within the specified node limit. Observe how a high (100,000) node limit produced fewer successful runs and incurred a higher search cost than learning with a lower (5000 or 10,000) limit.

Metareasoning can monitor the overall skill of a self-adaptive system with bounded rationality and use it to control the learner. When the system addresses a sequence of problems under some resource bound per problem, a coarse but significant measure of its success is simply how many problems it solves during its learning phase. FORR includes the option of *full restart*, the ability to reinitialize all weights to the same small value and begin again, on different problems from the same class (Petrovic and Epstein, 2006a). (This is different from repeated restart on the same problem, which is indented to diversify search for a single solution (Gomes and Sellman, 2004)). When too few problems have been solved during a learning phase, full restart begins the entire phase over again. Under a high node limit (e.g., 100,000 in Figure 6), full restart modulates the

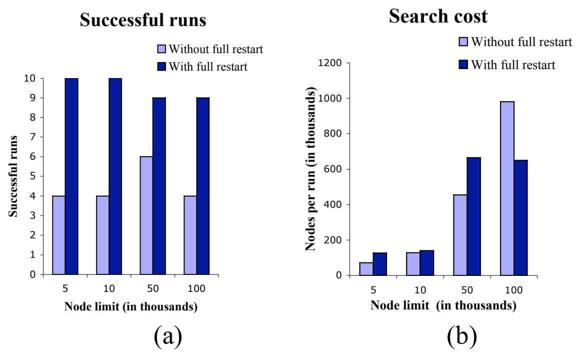

Figure 6: The impact of resource bounds on the geometric problems. (a) More runs in an experiment are successful with full restart (darker bars) and (b) fewer nodes are expanded.

heavy-tail difficulty. It abandons a run on an unusually difficult sample of problems from the class, or one where the problems were so easy that misinformation was acquired. The resource limit is crucial here, as Figure 6(a) indicates. As one would expect, higher resource bounds incur a higher learning cost, but full restart considerably modulates that effect.

Modeling Expertise

Metareasoning permits a system to assess its performance on an individual problem. To gauge how well a problem solver has done, however, requires some standard of expertise: an oracle, perhaps, or an expert opponent. Although such guidance is important, it may not offer enough variety to develop a robust learner.

Experiments with Hoyle, for example, found that playing against a *perfect player* (a program that always makes an optimal move) was too narrow (Epstein, 1994b). An expert game player, after all, should hold its own against opponents of any strength. When Hoyle trained against a perfect player, it later lost testing contests to opponents with far less prowess —it was repeatedly flummoxed by their errors. We experimented with many alternatives. The introduction of some percentage of random decisions into an otherwise flawless opponent drove the learner's experience outside the narrow realms of perfect play. Those random moves lacked good rationales, however, and were therefore often of low quality, not the kind of decisions Hoyle should learn to make. Self-training, where Hoyle played only against itself, did not always develop sufficiently strong expertise either. Our most effective approach, *lesson and practice training,* combined supervised and self-supervised learning: Hoyle alternately played 2 contests against a perfect player and then practiced in 7 contests against itself. Table 2 shows the results.

Without an external standard of expertise for a problem class, a system can take traces of its own successes as a model. Neither Ariadne nor ACE has an external model; each program learns alone, and can only judge the correctness of its actions from their ultimate result: a goal found or a CSP solved. When the robot finds the goal, metarea-

Table 2: Skill after learning tic-tac-toe with 2 different models of expertise. Lesson and practice training was better preparation for competition against opponents of different strengths at this draw game than training against a perfect player. Hoyle played 100 testing contests against each of 4 opponents: one moved perfectly, the others had some percentage of random moves among otherwise perfect play: 10% random (expert), 70% random (novice), and 100% (random).

Outcomes	***Perfect player***	***Lesson and practice***
Wins against an expert	12	18
Wins against a novice	59	63
Wins against random play	80	85
Draws against perfect play	100	100
Draws against an expert	93	98
Draws against a novice	80	97
Draws against random play	88	100

soning excises any closed loops from a trace of the robot's path, and takes the remainder as a model. Similarly, when ACE solves a problem, metareasoning excises any assignments that are subsequently retracted, and takes the remaining search tree as a model.

Unfortunately, neither a loop-free path nor a retraction-free search is likely to be ideal —there may have been a faster way to solve the problem. The temptation is to set some arbitrary standard for performance, and then insist upon it. The impact of bounded rationality is mixed, however, as Table 2 warned. Instead we increase the granularity of the metareasoning, as described in the next section.

Learning about Reasoning

Metareasoning permits a system to assess the performance of its individual components. Although FORR's structure is a 3-tier hierarchy, most Advisors are expected to lie in tier 3. Thus an obvious learning target in FORR is the class-appropriateness of each individual heuristic, represented as a weight for each tier-3 Advisor. FORR learns these weights and then uses them in tier 3's voting. (The relatively few Advisors in the other tiers are unweighted; they are consulted in some pre-specified order instead.)

Given j Advisors, a choice c is selected in tier 3 based on both the strength $s(A_j, c_i, C)$ that each Advisor A_j expresses for that choice and the weight w_j of the Advisor:

$$\underset{c \in choices}{\arg\max} \sum_{A_j \in Advisors} w_j s(A_j, c, C)$$

Ideally, only the best Advisors should participate in decision making, and those that offer better advice should be emphasized more. An irrelevant Advisor (e.g., Material in tic-tac-toe) simply fails to produce any advice. The meta-level can instruct the object level to omit such an Advisor from computation. For the others, FORR's metareasoning extracts *training instances* from the (likely imperfect) trace of a solved problem in the learning phase. A training instance is a problem state, the available choices there, and a decision made by the model of expertise. A *positive* training instance is a correct action selection; a *negative* training instance is an error. FORR's weight learning then judges the performance of Advisors on training examples.

Identifying Good Advice

Because FORR's tier-3 Advisors express their preference for choices by assigning them numerical strengths, a mechanism is needed to judge whether such a set of values is correct on the choice specified by a training instance. FORR offers two options: top-rated and relative support. Under *top-rated*, an Advisor is considered correct on a positive training instance only if it gives the decision a strength at least as high (low for negative training instances) as any other it assigned within the set of choices. Another way to judge correctness attends more carefully to the nuances of variation in the metric. The *relative support* $rs(A, c, C)$ of an Advisor A for choice c in a set of available choices C is the normalized difference between the strength the Advisor assigned to c and the average of the strengths the Advisor assigned to all the choices in C:

$$rs(A,c,C) = \frac{s(A,c,C) - avg(A,C)}{avg(A,C)}, avg(A,C) = \frac{\sum_{e \in C} s(A,e,C)}{|C|}$$

Under relative support, an Advisor A is considered correct on a training instance with decision $c \in C$ if and only if $rs(A, c, C)$ is positive.

Reasoning about Learning

Metareasoning permits a system to assess the significance of an individual training instance. Not all training instances are equally important. They are likely to be drawn, as discussed above, from problems of inherently different difficulty. Moreover, training instances may be of different kinds. For example, in CSP search, a decision selects either a variable to consider or a value to assign it. When variables are selected cleverly, propagation is particularly effective. Thus good variable selection makes value selection easier, and therefore less significant.

Weight learning in FORR reinforces an Advisor's weight with a *reward* (increment) or a *penalty* (decrement) based on its correctness on each training instance. FORR tallies the number of training instances on which each tier-3 Advisor gives the correct advice. Some Advisors, such as the fork detector in Hoyle, produce important advice, but rarely. Thus a tally is not enough. The fraction of times that an Advisor's advice has been correct is a somewhat better measure. ACE, however, required the considerably more sophisticated metareasoning of DWL and RSWL.

DWL (Digression-based Weight Learning) judges correctness with *top-rated* (Epstein, Freuder and Wallace, 2005). DWL calculates reinforcements in proportion to problem difficulty, gauged by the resources consumed to solve it. On training instances from relatively short solutions DWL assigns larger rewards to variable-selection Advisors than to value-selection Advisors. (DWL uses metareasoning to estimate "relatively short" based on its performance on earlier problems in the same learning phase.) DWL also reinforces behavior on a negative training example in proportion to the size of the *digression* (eventually abandoned search tree) it began.

RSWL (Relative Support Weight Learning) judges correctness with relative support (Petrovic and Epstein, 2006b). RSWL reinforcements are directly proportional to relative support, but penalties are also inversely proportional to the number of choices. Two variations on RSWL assign rewards and penalties based on their estimation of a training instance's difficulty. To gauge difficulty, RSWL-κ uses κ, and RSWL-d uses search tree depth. κ is a measure of constrainedness developed for CSP classes (Gent et al., 1996). Current search tree depth is considerably less expensive to compute than a dynamic value for κ. RSWL-d assumes that decisions are more difficult at the top of the search tree. (This is a reasonable assumption, given that every CSP has a *backdoor*, a set of variables after whose consistent assignment with the constraints search becomes extremely easy (Williams, Gomes and Selman, 2003).) The performance of all four weight-learning algorithms is com-

Table 3: Performance improves on two CSP classes after learning with different algorithms. For 50 testing problems each, space is nodes searched.

Learning algorithm	*Geometric* *Space*	*Geometric* *Solved*	*Composed* *Space*	*Composed* *Solved*
DWL	225.4	98.0%	298.5	95.4%
RSWL	237.1	98.6%	161.1	97.8%
RSWL-d	215.9	98.8%	208.0	96.6%
RSWL-κ	189.8	98.4%	218.1	96.6%

pared in Table 3.

Correct weights only compare Advisors, however; low-weighted heuristics are still likely to give poor advice. To identify the more class-appropriate Advisors, FORR uses *benchmark Advisors*, which produce advice at random. A benchmark Advisor does not participate in voting, but it does receive a learned weight. After the learning phase, metareasoning eliminates from participation any Advisor whose weight is lower than its benchmark's. As an added benefit, any representation that was referenced only by the eliminated Advisors will no longer be computed. Filtering the Advisors in all these ways speeds decisions after learning, without decreasing performance. In ACE, for example, the average testing decision is accelerated by about 30%.

Learning Competent Reasons

Learning on a sequence of problems applies knowledge from one successful search to subsequent searches. The first success increases the weights of those Advisors that contributed to its decisions. It may, however, be quite expensive to solve any first problem at all, particularly when the problems are hard, there are many Advisors (perhaps as duals), and they disagree with one another. Under bounded resources, only the easiest problems in a class may be solved. There will be relatively few training instances and they will all be drawn from relatively easy situations. Indeed, on occasion a run fails because the learner has solved no problems at all, and therefore changed no weights.

FORR's metareasoning therefore includes the ability to work with *random subsets* of Advisors (Petrovic and Epstein, 2007). For each new problem in the learning phase, this method chooses a subset of tier-3 Advisors to consult; if search solves that problem, FORR learns weights from it only for that subset of Advisors. The expectation is that eventually the subset chosen for some problem will be dominated by class-appropriate heuristics, the problem will be solved, the class-appropriate weights in the subset will increase, and whenever any of those Advisors appears in a subset for a subsequent problem it will be more likely to influence search, making further successes more likely. Subset size is crucial, however: it must be large enough to expose Advisors to learning frequently, yet small enough to speed processing and to give an otherwise minority voice the opportunity to dominate decisions. Table 4 shows how performance improves using random subsets with full restart. Learning with random subsets reduces the number of problems addressed during learning and speeds computation time on each decision. It even functions well when we deliberately skew the initial Advisor pool with many more class-inappropriate than class-appropriate Advisors.

Table 4: Random subsets of 30% improve ACE's learning performance on the geometric problems with full restart, without a statistically significantly change in testing performance. An early failure is an unsolved problem before any solved ones.

	Random subsets *Without*	*Random subsets* *With*
Learning problems	44.8	36.1
Learning failures	13.7	6.4
Early learning failures	7.1	0.9
Successful runs (of 10)	10	10
Time per learning decision	0.0161	0.0106
Time per learning run	1651.6	593.6
Average nodes in testing	192.7	195.9
Solved testing problems	98.6%	96.4%

Learning to Stop Learning

A self-aware system that gauges its own overall skill can recognize the rate at which its performance improves. If it is no longer learning anything new, it should stop learning. Metareasoning to stop learning is implemented in FORR as *learning to stability* (Epstein, Freuder and Wallace, 2005). Under this option, FORR monitors its Advisors' weights across a recent time window (e.g., the last 20 problems) and terminates a learning phase when the Advisors' weights are no longer changing appreciably (the standard deviation of changes in them across the window are less than ε). Learning to stability assumes that stable weights will remain stable; our experience over far longer learning phases confirms this. Making a learner more responsive to its own learning experience this way has proved successful in all three domains: there is no change in performance, only a reduction in the resources that would have been devoted to learning after the system found no further way to improve its weights. For example, Hoyle recognizes that it has learned all it can on simple games after 12 contests; more difficult games require as many as 120.

Learning to Reason Less

In domains where errors are not fatal, more thinking is not always better. (Such economy must of course be evaluated by the risks of error it presents and the cost required to recover from such errors.) Metareasoning can monitor the traces from individual problems to reduce computation in a variety of ways, thereby restructuring the reasoning process itself. We discuss several such approaches here.

One might think that a tier-3 Advisor with a particularly high weight could be promoted to the end of the tier-1 list, where it could provide guidance earlier and save the resources otherwise directed to tiers 2 and 3. In our experience in all three domains, however, this is a dangerous practice. When we tested it, even with an Advisor whose weight was dramatically higher than the others, Hoyle lost some contests against strong players, Ariadne never formulated the plans that would have supported its best paths, and ACE made rare, but disastrous errors that produced extremely large digressions. Simply put, a heuristic is unlikely to be always right. That is why FORR groups them together and relegates them to tier 3.

Tier-3 Advisors retained after weight learning give better than random advice, but they may not be uniformly appropriate. Under *prioritization*, FORR partitions tier 3 after learning. In this scenario, Advisors with the highest weights vote first; only if there is a tie do subsequent groups of Advisors have an opportunity to comment. The advantage is that fewer resources are likely to be consumed, since ties are relatively rare after the first subset or two. The nature and granularity of the partition is important, however. Because fixed-size partitions ignore natural cutoffs, FORR's partitioning method places tier-3 Advisors into groups of uneven size, based on their weights. If there are too many groups, prioritization effectively produces a ranked list. (Ranking underperforms a weighted mixture in all three of our domains, for every problem class we have investigated.) Partitions of 3 to 7 subsets produce the best results, but the number of subsets depends on the problem class. In any case, we rarely experience more than a 10% speedup with prioritization. Fewer Advisors make more mistakes, and most representations are still computed, particularly the computationally intensive ones, on which the most class-appropriate Advisors rely.

Metareasoning can identify portions of the solution process where different behavior is warranted. In some domains, the last part of problem solving is more formulaic. Game players often have an endgame library, and play by lookup (e.g., Chinook, (Schaeffer et al., 2005)); there is no need for advice at that point, only rote play. The last part of a CSP occurs after its backdoor. Metareasoning estimates an upper bound on the expected size of the backdoor as the maximum search depth at which it has experienced a wipeout within the problem class. Under an option called *Pusher*, below the maximum wipeout depth ACE consults the single highest-weighted tier-3 variable-selection Advisor as if it were in tier-1 (Epstein, Freuder and Wallace, 2005). In the event of a tie, Pusher chooses a variable lexically, bypassing tier 2 and tier 3 entirely. Pushing generally reduces computation time by about 8%. ACE does not, however, push value selection. Experiments indicated that one can think less about where to search after the backdoor but that thinking more about the values to assign there is still worthwhile.

Fast and frugal reasoning is a form of human metareasoning that favors recognized choices and then breaks ties among them with a single heuristic (Gigerenzer, Todd and Group, 1999). For ACE, a recognized choice is one made earlier in search (and subsequently retracted) on the same problem. We tested several strategies (random, most recently used, highest weighted) to select the single heuristic. None ever harmed performance. Moreover, on more difficult problems, where retractions are more common, reusing prior decisions with the highest weighted Advisor to break ties accelerated decision time, despite increased errors (Epstein and Ligorio, 2004).

Discussion

On difficult problems, errors in a model of expertise may be inevitable, and training examples from the same model may vary in their quality and significance. Nonetheless, a self-aware system can recognize its own prowess or lack thereof, and respond accordingly. Moreover, as we have shown here, a self-aware system can evaluate and reorganize its components to improve its performance.

ACE learns to solve problems in many difficult classes, problems that stymie off-the-shelf solvers (like those in Table 1) without the ability to monitor and modify their own behavior. Constraint solving is a paradigm for many kinds of difficult problems. Hoyle and Ariadne each learn how to search a single space, a game tree or a maze, from which all the problems in a class are drawn. ACE learns about how to search a *set* of spaces, all of which are supposedly alike, a considerably more difficult task.

Much remains to be done. Hoyle learns from lost contests, but Ariadne and ACE do not yet learn from failure. (ACE can, however, learn in a problem class with both solvable and unsolvable problems, and successfully apply its learned knowledge to both.) There are typically very few Advisors in tier 1; even a novice in the domain can readily prespecify an order for them. Advisors in tier 2, however, produce plans, and how to order them is less obvious. Weight learning for tier 2 is future work. Finally, the metareasoning described here still depends on settings for some parameters, particularly resource limits, the restart threshold, and random subset size. Future work includes an investigation of the ways those values interact with one another. Factor analysis, for example, has indicated that many CSP heuristics produce similar advice even from very different viewpoints (Wallace, 2006). FORR should capitalize on that.

Metareasoning is essential in FORR's ability to learn to solve problems within a given class. As it learns a combination of heuristics, FORR uses metareasoning to decide when to abandon an unpromising learning attempt (full restart), when to stop learning (the stability criterion), how to select heuristics during learning (weights and random subsets), and how to prioritize heuristics. FORR also reasons about its performance on previous problems (DWL), its previous decisions (fast and frugal reasoning), and the relative discriminatory power of its heuristics (RSWL).

Acknowledgements

This work was supported by the National Science Foundation under 9222720, IRI-9703475, IIS-0328743, and IIS-0739122. ACE is an ongoing joint project with Eugene Freuder and Richard Wallace. Joanna Lesniak, Tiziana Ligorio, Xingjian Li, Esther Lock, Anton Morozov, Barry Schiffman, and Zhijun Zhang have all made substantial contributions to this work.

References

Aardal, K. I., S. P. M. van Hoesel, A. M. C. A. Koster, C. Mannino and A. Sassano 2003. Models and solution techniques for frequency assignment problems. 4OR: A Quarterly Journal of Operations Research 1(4): 261-317.

Biswas, G., S. Goldman, D. Fisher, B. Bhuva and G. Glewwe 1995. Assessing Design Activity in Complex CMOS Circuit Design. *Cognitively Diagnostic Assessment*. Nichols, P., S. Chipman and R. Brennan. Hillsdale, NJ, Lawrence Erlbaum**:** 167-188.

Borrett, J., E. P. K. Tsang and N. R. Walsh 1996. Adaptive Constraint Satisfaction: the Quickest First Principle. In *Proc. of 12th European Conference on AI*, 160-164.

Cox, M. T. and A. Raja 2007. Metareasoning: A Manifesto, Technical Report. , BBN Technologies.

Crowley, K. and R. S. Siegler 1993. Flexible Strategy Use in Young Children's Tic-Tac-Toe. *Cognitive Science* 17(4): 531-561.

D'Andrade, R. G. 1991. Culturally Based Reasoning. *Cognition and Social Worlds*. Gellatly, A. and D. Rogers. Oxford, Clarendon Press**:** 795-830.

Epstein, S. L. 1994a. For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science* 18(3): 479-511.

Epstein, S. L. 1994b. Toward an Ideal Trainer. *Machine Learning* 15(3): 251-277.

Epstein, S. L. 1998. Pragmatic Navigation: Reactivity, Heuristics, and Search. *Artificial Intelligence* 100(1-2): 275-322.

Epstein, S. L. 2001. Learning to Play Expertly: A Tutorial on Hoyle. *Machines That Learn to Play Games*. Fürnkranz, J. and M. Kubat. Huntington, NY, Nova Science**:** 153-178.

Epstein, S. L., E. C. Freuder and R. J. Wallace 2005. Learning to Support Constraint Programmers. *Computational Intelligence* 21(4): 337-371.

Epstein, S. L. and T. Ligorio 2004. Fast and Frugal Reasoning Enhances a Solver for Really Hard Problems. In *Proc. of Cognitive Science 2004*, 351-356. Chicago, Lawrence Earlbaum.

Fukunaga, A. S. 2002. Automated Discovery of Composite SAT Variable-Selection Heuristics. In *Proc. of AAAI/IAAI*, 641-648.

Gagliolo, M. and J. Schmidhuber 2007. Learning dynamic algorithm portfolios. *Annals of Mathematics and Artificial Intelligence* 47(3-4): 295-328.

Gent, I. E., E. MacIntyre, P. Prosser and T. Walsh 1999. The Constrainedness of Search. In *Proc. of Thirteenth National Conference on Artificial Intelligence*, 246-252.

Gigerenzer, G., P. M. Todd and A. R. Group 1999. *Simple Heuristics that Make Us Smart*. New York, Oxford University Press.

Gomes, C. P. and M. Sellman 2004. Streamlined Constraint Reasoning. In *Proc. of CP-2004*, 274-289. Springer.

Gomes, C. P. and B. Selman 2001. Algorithm portfolios. *Artificial Intelligence* 126(1-2): 43-62.

Gomes, C. P., B. Selman, N. Crato and H. Kautz 2000. Heavy-tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning* 24: 67-100.

Johnson, D. B., C. R. Aragon, L. A. McGeooh and C. Schevon 1989a. Optimization by Simulated Annealing: An experimental evaluation; Part 1, Graph partitioning. *Operations Research* 37: 865-892.

Johnson, D. S., C. R. Aragon, L. A. McGeoch and C. Schevon 1989b. Optimization by Simulated Annealing: An Experimental Evaluation; Part
I, Graph Partitioning. *Operations Research* 37: 865-892.

Keim, G. A., N. M. Shazeer, M. L. Littman, S. Agarwal, C. M. Cheves, J. Fitzgerald, J. Grosland, F. Jiang, S. Pollard and K. Weinmeister 1999. PROVERB: The Probabilistic Cruciverbalist. In *Proc. of Sixteenth National Conference on Artificial Intelligence*, 710-717. Orlando, AAAI Press.

Minton, S., J. A. Allen, S. Wolfe and A. Philpot 1995. An Overview of Learning in the Multi-TAC System. In *Proc. of First International Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, Oregon, USA.

Nareyek, A. 2003. Choosing Search Heuristics by Non-stationary Reinforcement Learning. *Metaheuristics: Computer Decision-Making*. Resende, M. G. C. and J. P. deSousa. Boston, Kluwer**:** 523-544.

Petrovic, S. and S. L. Epstein 2006a. Full Restart Speeds Learning. In *Proc. of FLAIRS-2006*.

Petrovic, S. and S. L. Epstein 2006b. Learning Weights for Heuristics that Solve Constraint Problems. In *Proc. of Workshop on Learning to Search at AAAI-2006*, 115-122. Boston.

Petrovic, S. and S. L. Epstein 2007. Random Subsets Support Learning a Mixture of Heuristics. In *Proc. of FLAIRS 2007*, Key West, AAAI.

Ratterman, M. J. and S. L. Epstein 1995. Skilled like a Person: A Comparison of Human and Computer Game Playing. In *Proc. of Seventeenth Annual Conference of the Cognitive Science Society*, 709-714. Pittsburgh, Lawrence Erlbaum Associates.

Schaeffer, J., Y. Bjornsson, N. Burch, A. Kishimoto, M. Muller, R. Lake, P. Lu and S. Sutphen 2005. Solving Checkers. In *Proc. of IJCAI-05*, 292-297.

Schraagen, J. M. 1993. How Experts Solve a Novel Problem in Experimental Design. *Cognitive Science* 17(2): 285-309.

Streeter, M., D. Golovin and S. F. Smith 2007. Combining multiple heuristics online. In *Proc. of AAAI-07*, 1197-1203.

Wallace, R. J. 2006. Analysis of heuristic synergies. Recent Advances in Constraints. Joint ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming - CSCLP 2005, LNCS 3978. Carlsson, M., F. Fages, B. Hnich and F. Rossi. Berlin, Springer.

Williams, R., C. Gomes and B. Selman 2003. On the Connections between Heavy-tails, Backdoors, and Restarts in Combinatorial search. In *Proc. of SAT 2003*.