Metareasoning: Thinking about Thinking

Michael T. Cox (ed.), Anita Raja (ed.)

https://doi.org/10.7551/mitpress/9780262014809.001.0001

Published online: 22 August 2013 **Published in print:** 18 March 2011 **Online ISBN:** 9780262295284 **Print ISBN:** 9780262014809

CHAPTER

4 Learning Expertise with Bounded Rationality and Self-Awareness

Susan L. Epstein, Smiljana Petrovic

https://doi.org/10.7551/mitpress/9780262014809.003.0004

Published: March 2011

Abstract

This chapter describes how an architecture enhanced by metareasoning assesses expertise, manages large bodies of heuristics, and learns to think more effectively. FORR (FOr the Right Reasons) is a learning and problem-solving architecture that models the development of expertise with bounded rationality and self-awareness. Metareasoning is essential in FORR's ability to learn to solve problems within a given class. As it learns to manage heuristics, FORR uses metareasoning to decide when to abandon an unpromising learning attempt, when to stop learning, how to select heuristics during learning, and how to prioritize heuristics. FORR also reasons about its performance on previous problems, its previous decisions, and the relative discriminatory power of its heuristics. Thinking about thinking makes FORR-based applications more incisive and more successful.

Keywords: metareasoning, FORR, expertise, bounded rationality, self-awareness, problem solving architecture

Subject: Artificial Intelligence

When people provide only rudimentary knowledge about a problem domain, a *self-adaptive* system must learn its own expertise, that is, become better and faster at its task than the rest of us (D'Andrade, 1991). Essential to that development are *bounded rationality* (imposed resource limitations) and *self-awareness*, the ability to monitor one's own problem-solving behavior and reasoning. From this perspective, *metareasoning* examines and reflects upon the components of decision making and how they are controlled, rather than merely upon the decisions themselves. *Learning with metareasoning* occurs when the system seeks to improve its performance by the application of metareasoning to modify its behavior. A system with this capacity could, for example, decide to ignore some of its experience, to prefer some procedures to others, or even to stop learning because it is satisfied with what it has accomplished. Under the aegis of an architecture that supports learning

with metareasoning, the three ambitious programs described here develop considerable expertise from experience with relatively few problems.

Despite the computational difficulties discussed by Conitzer (this vol., chap. 8), metareasoning often successfully enhances object-level reasoning. For example, Arcos, Mülâyim, and Leake (this vol., chap. 11) describe how introspective reasoning can improve a case-based system, and Alexander, Raja, and Musliner (this vol., chap. 5) describe how metalevel reasoning can successfully limit the resources for scheduling based on a Markov decision process. Zilberstein (this vol., chap. 3) examines the relationship between optimal metareasoning and bounded rationality on the object level. This chapter describes how an architecture enhanced by metareasoning assesses expertise, manages large bodies of heuristics, and learns to think more effectively.

The Architecture and the Problems

FORR (FOr the Right Reasons) is a learning and problem-solving architecture that models the development of expertise with bounded rationality and self-awareness (Epstein, 1994a). Figure 4.1 describes FORR as an
p. 44 elaboration on the fundamental ↳ metareasoning diagram in figure 1.2 of chapter 1 of this volume. At the domaindependent ground level, FORR describes each world state as it appears during search for a solution, and executes actions to move from one state to the next. At the object level, FORR re-represents the current state and reasons about it to select the next action in a sequence of decisions that addresses a single task. Here, decision makers are reactive procedures, opportunistic planners, or heuristics. After each task, the object-level forwards to the metareasoning level the history and outcome of the task, the computational resources consumed, and the contribution of each heuristic to each decision.

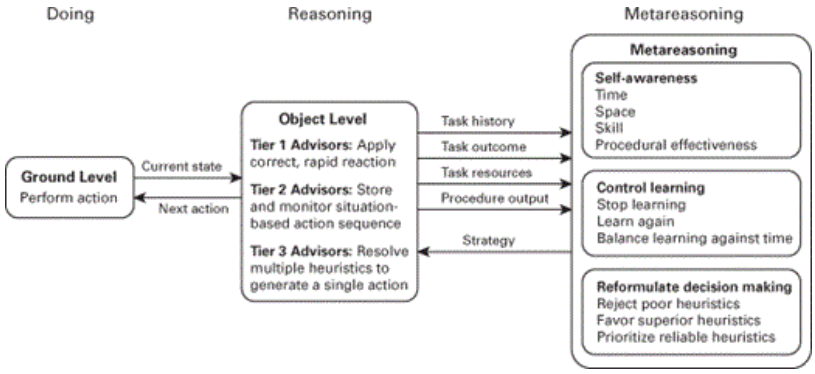

Figure 4.1 FORR in the context of chapter 1.

FORR addresses a *problem class* (a set of similar problems). Metareasoning assesses the program's skill within that class and the effectiveness of each procedure there. Metareasoning also determines whether to continue learning about the class, to stop, or to restart the entire learning process. Most dramatically, for a given class, metarea-soning can reformulate the object level to eliminate ineffective procedures and favor superior ones. This chapter draws on three very different challenges addressed by FORR-based programs, each of which can be solved with a sequence of actions: game playing, path finding, and constraint satisfaction.

Hoyle learns to play 19 two-person, perfect-information, finite-board games (Epstein, 2001). Here a problem class is a game (e.g., tic-tac-toe), and a problem is a single contest whose actions are legal moves. A solution is a
p. 45 move sequence that achieves ↳ an *ideal outcome* (a win or a tie, as defined by the game tree). Given the rules, Hoyle plays against itself or an external opponent. Some games in Hoyle's repertoire have game trees of several

billion nodes. The expert programs crafted as opponents required considerable game-specific knowledge and skill.

Ariadne learns to find paths in a two-dimensional grid maze (Epstein, 1998). Here a problem class is a maze, and a problem is a trip from an initial location to a goal. (The initial location and the goal vary from one problem to the next.) Ariadne has no map, only the coordinates of the robot and the goal, plus a view restricted to the grid positions between the robot and the nearest obstruction in four orthogonal directions. An action moves the robot in a straight line to a currently visible location. A solution is an action sequence that reaches the goal. Represented as a (cyclic) graph, a moderately challenging 20 × 20 maze would have 280 nodes and about 1,485 edges.

ACE (the Adaptive Constraint Engine) learns to solve constraint satisfaction problems (*CSP*s) (Epstein, Freuder, & Wallace, 2005). Here, a problem is a (binary) CSP: a set of variables, each with an associated set of values (*domain*), and a set of constraints that restrict how pairs of variables can be bound to values simultaneously. A CSP can be represented as a *constraint graph* with a vertex for each variable and an edge for each constraint. A problem class is a set of CSPs with the same descriptive characterization (e.g., number of variables and maximum domain size). Each action in ACE either selects a variable or assigns a value to a selected variable from its associated domain. After each assignment, *propagation* infers the impact on the domains of the as-yet-unassigned variables. If any variable then has no values consistent with the current assignments (*wipeout*), the most recent assignments are retracted chronologically until every variable has some possible value. A solution is an assignment sequence that satisfies all constraints and assigns every variable a value. (Although traditionally a CSP solution is merely an acceptable assignment to all the variables, some assignment sequences are more efficient than others, in terms of the inference they require. Expertise here values efficiency as well as the ability to find a valid assignment.) CSPs model many of the NP-hard problems addressed by AI systems, including graph coloring, propositional satisfiability, and scheduling.

Structured CSPs are more similar to real-world problems than simple random ones. Structured problems' pockets of densely connected variables with very tight constraints often present particular challenges for traditional solvers. Indeed, the opposites of traditional heuristics often fare better on them than the originals. We report here on a composed class (Aardal et al., 2003) with 30 variables and domain size 6 (7^{30} search nodes) and a geometric class (Johnson et al., 1989) with 50 variables and domain size 10 (11^{50} nodes). Problems from both classes often stumped the best solvers at a recent competition.

p. 46

Foundation Assumptions

The following premises dictate FORR's general structure and behavior:

- *Good reasons underlie intelligent actions*. A good reason is a justification for a decision. Domain-specific good reasons are called *Advisors*. Input to any Advisor is the current problem state and the available legal actions. Output from any Advisor is *advice* about some number of actions. For example, *Material* advises Hoyle to capture an opponent's piece.

- *A domain has many such good reasons*. Human experts typically rely on a host of domain-specific Advisors. ACE, for example, has more than 100 Advisors drawn from the constraint literature.

- *Some good reasons are always correct*. These *tier-1 Advisors* take priority during decision making. Their fast, accurate guidance mandates an action or eliminates it from further consideration. For example, Ariadne's *Victory* always moves directly to a visible goal.

- *Some good decisions include several actions*. These *tier-2 Advisors* 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. At most one plan is active at a time. The tier-1 Advisor *Enforcer* appropriately supports the execution of an active plan or terminates it.

- *Most good reasons are fallible heuristics*. These *tier-3 Advisors* express preferences for choices numerically, as *strengths*. For example, Ariadne's *Giant Step* assigns greater strengths to longer steps. The object level in figure 4.1 coordinates all three tiers of Advisors to make a decision. It progresses to the next tier only if the current one does not produce a decision.

- *Good reasons make different contributions on different problem classes*. FORR evaluates each tier-3 Advisor on each new problem class. The more constructive are *class-appropriate*; the others are *class-inappropriate*. ACE's heuristics, for example, are provided in *dual pairs* (one to maximize and the other to minimize the same metric); ACE sorts out whether either is appropriate. Lines 1 and 2 in table 4.1 show that duals may perform differently 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 & Siegler, 1993; Ratterman & Epstein, 1995; Schraagen, 1993) and in programs that integrate multiple rationales to their advantage (Keim et al., 1999). Table 4.1 (lines 3–5) shows how a pair 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 4.1 (lines 3 and 5) shows that pairs of
p. 47 Advisors successful on one problem class may do less well on another. ↳

- *A program can* learn *to become an expert*. In the *learning phase*, FORR addresses a sequence of problems, and all of figure 4.1 is active. Metareasoning examines the data from the object level after each learning-phase problem, and may reformulate its strategy. Then, in the *testing phase*, FORR turns learning off and evaluates its performance on a second set of problems. No further directives from the metareasoning module occur during testing. Because decisions may be nondeterministic, FORR evaluates performance over a set of such runs (an *experiment)*. Relatively few learning problems should suffice. For example, ACE learns to solve each problem class well after learning on only 30 problems. Table 4.1 (line 6) shows how the mixture of heuristics that ACE learned for each of them outperformed both individual heuristics and pairs of them.

- *Multiple representations enhance reasoning*. Each Advisor has access to every 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, ACE has dozens of CSP descriptions, including many that identify different kinds of subproblems and relationships among variables.

Table 4.1 A heuristic's appropriateness varies with the problem class, and multiple heuristics outperform individual ones in CSP search. Variable-ordering metric *ddd* is the ratio of a variable's dynamic domain size to the number of its unassigned neighbors in the constraint graph. v_1 *and* v_2 are value-ordering heuristics. After propagation, v_1 prefers values that produce the smallest domain size, and v_2 prefers values that produce the largest product of domain sizes. Space is nodes searched; elapsed time is in CPU seconds. Fifty problems were tested in each class.

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

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, & Walsh, 1996). Heuristics from a *portfolio* may be selected to compete in parallel until one solves the problem, or may be used in turn on the same problem (Gagliolo & Schmidhuber, 2006; Gomes & Selman, 2001; Streeter, Golovin, & Smith, 2007). A system may also label its heuristics individually for their appropriateness with learned *weights*. Heuristics can be ranked by
p. 48 weight and consulted one at a time (Minton et al., 1995; ↳ Nareyek, 2003) or they can *vote*, combining advice from each of them on every decision (Fukunaga, 2002). In FORR, tier-1 and tier-2 Advisors are consulted in some prespecified, class-independent order; in tier 3, Advisors vote.

Learning with Bounded Rationality

Despite a fixed characterization, a random problem generator may not create a uniformly difficult problem class. For example, the difficulty a fixed algorithm experiences within a class of putatively similar, randomly generated CSPs may decrease according to a power law (Gomes et al., 2000). As a result, there are many far more difficult problems within the same class.

Bounded rationality sets an arbitrary performance standard. One might, for example, learn only from solutions achieved within bounded resources. This presumes that a fast solution or one that visits fewer search states is better. Setting such bounds is problematic, however. Under bounds too low for typical problems in the class, only the easiest are solved, solutions are fewer, and the learned behavior is untrustworthy because it arose from problems where most any decision would do. If bounds are set too high, however, long but successful searches provide traces of not-so-expert behavior. For example, figure 4.2 shows the impact on ACE of a limit on *search nodes* (partial assignments) while it learns to solve the geometric problems. A run was *successful* if it solved at least 80 percent of its 50 testing problems within the node limit. Observe how higher node limits during learning produced fewer successful runs and incurred a higher search cost than did lower limits.

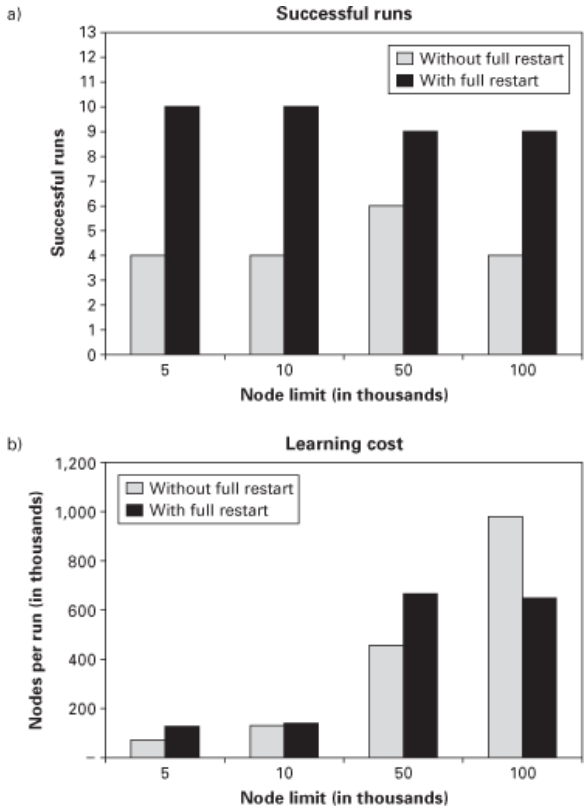

Figure 4.2 The impact of resource bounds on 10 runs for the geometric problems. (a) More runs in an experiment are successful with full restart (darker bars) and (b) fewer nodes are expanded.

Metareasoning can monitor overall skill and use it to control the learner. A coarse but significant measure of success is the frequency of solved problems during learning. FORR introduces *full restart*, the ability to reinitialize all heuristics' weights to some small value and begin again, on different problems from the same class (Petrovic & Epstein, 2006a). (This is different from restart on a problem, which tries to diversify search for its solution [Gomes & Sellman, 2004]). When problems frequently go unsolved during a learning phase, full restart begins the entire phase again. This abandons a run on an unusually difficult sample of problems, or one where solutions to very easy problems introduce misinformation. The resource limit is crucial here, as figure 4.2 indicates. As one would expect, higher resource bounds incur a higher learning cost, but under a high node limit, full restart considerably modulates that effect.

Modeling Expertise

Modeling expertise requires some standard: an oracle, perhaps, or an expert opponent. Such guidance,
p. 49 although important, may not offer enough variety to develop a robust ↳

p. 50 ↳
learner. Experiments with Hoyle in table 4.2, for example, found competition against a *perfect player* (a program that makes only optimal moves) too narrow (Epstein, 1994b). An expert player should succeed against opponents of any strength. After training against a perfect player, Hoyle lost testing contests to opponents with far less prowess—it was repeatedly flummoxed by their errors. Although some percentage of random decisions in an otherwise flawless opponent drives the learner's experience outside perfect play, such moves lack good rationales and are therefore often of poor quality—not the kind of decisions Hoyle should learn to make. If Hoyle learned only against itself, it did not always develop sufficiently strong expertise either. The most effective approach, *lesson and practice training*, had Hoyle alternately play two contests against a perfect player and then practice in seven contests against itself.

Table 4.2 Skill after learning against different playing expertise (Epstein, 1995). For competition against opponents of different strengths at this draw game, lesson and practice training was better preparation 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 percent random (expert), 70 percent random (novice), and 100 percent (random).

	Percentage of contests played correctly	
Outcomes	**After learning against a perfect player**	**After lesson and practice training**
Wins against an expert	12%	18%
Wins against a novice	59%	63%
Wins against random play	80%	85%
Wins or draws against perfect play	100%	100%
Wins or draws against an expert	93%	98%
Wins or draws against a novice	80%	97%
Wins or draws against random play	88%	100%

Without an external standard of expertise, a system can take traces of its own successes as a model. Neither Ariadne nor ACE has an external model; each learns alone and can only judge the correctness of its actions from its ability to solve problems. When the robot finds the goal, metareasoning 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 decisions that have no alternative or are subsequently retracted and takes the remaining ones as a model. Nonetheless, neither a loop-free path nor a retraction-free search is likely to be ideal—there may have been a better way to solve the problem.

p. 51

Learning about Advisors

Metareasoning permits a system to assess the performance of its individual components. Because most Advisors lie in tier 3, class-appropriateness, represented as a weight for each tier-3 Advisor, is an obvious learning target. FORR learns such weights and then uses them in tier 3's voting. Support for choice *c* is based on both the strength $s(A_j, c, C)$ that each Advisor A_j expresses for that choice among available choices *C* and the weight w_j of the Advisor. The choice with the highest support is selected:

$$\arg\max_{c \in C} \sum_{A_j \in Advisors} w_j s\left(A_j, c, C\right)$$

FORR's metareasoning extracts training instances from the (likely imperfect) trace of each solved problem during learning. A *training instance* is a problem state, the available choices there, and the decision made by the model of expertise. A *positive* training instance selects a correct action; a *negative* training instance does not. Weight learning judges the performance of Advisors on training instances.

Of course, training instances are not all equally important. They may be drawn from problems of inherently different difficulty, or be of different kinds. For example, in CSP search, a decision selects either a variable to consider or a value to assign it. Clever variable selection makes propagation particularly effective, and therefore makes value selection both easier and less significant.

To judge the correctness of advice labeled with numerical strengths, FORR has two options: top-rated strategies and relative support strategies. 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 (for negative training instances, as low) as any other it assigned there. Alternatively, the *relative support* $rs(A, c, C)$ of Advisor *A* for choice *c* in available choices *C* is the normalized difference between the strength *A* assigned to *c* and the average of the strengths *A* assigned across all of *C*:

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

Under relative support, which attends more carefully to the nuances of variation in an Advisor's output, *A* is considered correct on decision $c \in C$ if and only if $rs(A, c, C)$ is positive.

FORR reinforces an Advisor's weight with a *reward* (increment) or a *penalty* (decrement) based on its correctness on each training instance. FORR tracks the number of training instances on which each tier-3 Advisor gives the correct advice. A tally is not enough because some Advisors (e.g., Hoyle's fork detector) produce crucial advice, but rarely. The fraction of times that an Advisor's advice has been correct is a somewhat
p. 52 better measure. ACE, however, has required the considerably more sophisticated ↳ metareasoning of *DWL* (Digression-based Weight Learning) and *RSWL* (Relative Support Weight Learning).

DWL judges correctness with top-rated. It calculates reinforcements in proportion to problem difficulty, as gauged by the resources consumed to solve it, relative to those required to solve problems earlier in the same learning phase (Epstein, Freuder, & Wallace, 2005). On training instances from shorter solutions, DWL assigns variable-selection Advisors larger rewards but assigns value-selection Advisors smaller rewards. DWL also reinforces behavior on a negative training instance in proportion to the size of the *digression* (eventually abandoned search tree) it began.

RSWL judges correctness with relative support (Petrovic & Epstein, 2006b). Under RSWL, reinforcements are directly proportional to relative support, but penalties are also inversely proportional to the number of choices. Two variations on RSWL modify rewards and penalties based on different estimates of training instance difficulty. RSWL-κ uses *constrainedness* (a measure of a CSP class difficulty (Gent, Prosser, & Walsh, 1996) to estimate the difficulty of each search state dynamically. RSWL-d is computationally less expensive; it uses search tree depth and assumes that decisions are more difficult at the top of the search tree. RSWL-d is based on the premise that, with respect to a given algorithm, every CSP has a *backdoor*, a set of variables after whose consistent assignment with the constraints search becomes extremely easy (Williams, Gomes, & Selman, 2003).

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, and they disagree with one another. On occasion a run fails because the learner has solved no problems at all, and therefore changed no weights. Otherwise, only the easiest problems in a class may be solved, which produces relatively few training instances, all from relatively easy situations. FORR's metareasoning therefore includes the ability to work with *random subsets* of Advisors (Petrovic & Epstein, 2008). For each problem in the learning phase, this method chooses a fresh subset of tier-3 Advisors to consult; if search solves that problem, FORR learns weights from it only for that subset. 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 subsequent subset it will be more likely to influence search, making further successes more likely. Subset size is crucial, however: it must be large enough to select Advisors frequently, yet small enough to speed processing and to give an otherwise minority voice the opportunity to dominate decisions. Table 4.3 shows how performance improves using random subsets: It reduces the number of problems addressed during learning and speeds computation time
p. 53 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 4.3 Random subsets of 30 percent improve ACE's learning performance on the geometric problems with full restart under a 5,000-node limit, with no statistically significant change in testing performance. An early failure is an unsolved problem before any solved one.

	Learning	
	Without random subsets	**With random subsets**
Number of learning problems	44.8	36.1
Number of learning failures	13.7	6.4
Number of early (learning) failures	7.1	0.9
Number of successful runs (of 10)	10	10
Seconds per learning decision	0.0161	0.0106
Seconds per learning run	1651.6	593.6
Average number of nodes in testing	192.7	195.9
Percentage of solved testing problems	98.6%	96.4%

Finally, a self-aware system can monitor the rate at which its performance improves and stop learning once it is no longer learning anything new. FORR's metareasoning includes such *learning to stability* (Epstein, Freuder, & 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 standard deviation of changes in the Advisors' weights 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 way to improve its weights further. 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

Metareasoning can monitor the traces from individual problems to reduce computation in a variety of ways, thereby restructuring the reasoning process itself. Unless errors can prove fatal, more thinking is not always better. Of course, such economy should be evaluated against the risks error presents and the cost required to recover from it.

p. 54 Ideally, decision making uses only the best Advisors and emphasizes those that offer better advice. The metalevel can instruct the object level to omit from computation any Advisors that produce no advice during learning (e.g., Material in tic-tac-toe). Moreover, once weights are learned, FORR uses *benchmark Advisors*, which produce random advice, to identify class-appropriate Advisors. A benchmark Advisor does not vote, 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. Moreover, any representation referenced only by eliminated Advisors will no longer be computed. Filtering the Advisors in these ways speeds decisions after learning, without decreasing performance. In ACE, for example, the average testing decision is accelerated by about 30 percent.

Although a tier-3 Advisor with a particularly high weight could be promoted to tier 1, our experience in all three domains has found this a dangerous practice. Instead, FORR partitions tier-3 Advisors retained after weight learning into groups of uneven size, based on their weights. Under this *prioritization*, tier-3 Advisors with the highest weights vote first; only under a tie do subsequent groups of Advisors have an opportunity to comment on the best tied actions. Fewer resources are consumed this way, since ties are relatively rare after the first group or two. With 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 three to seven subsets produce the best results, but the number of subsets depends on the problem class. We have rarely experienced more than a 10 percent speedup with prioritization. Although fewer Advisors make more mistakes, most representations are still computed, particularly the computationally intensive ones on which class-appropriate Advisors often 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, for example, may have an endgame library and play there by lookup. ACE estimates the last part of CSP search as that after the maximum search depth at which it has experienced a wipeout within the problem class. Below this depth and only during testing, ACE's tier-1 Advisor *Pusher* consults the single highest-weighted tier-3 variable-selection Advisor as if it were in tier 1, bypassing tiers 2 and tier 3 entirely (Epstein, Freuder, & Wallace, 2005). Pushing generally reduces computation time by about 8 percent. ACE does not, however, push value selection. Experiments indicate that one can think less about where to search after the backdoor but not about the values to assign there.

Fast and frugal reasoning is a form of human metareasoning that favors recognized choices and then breaks ties among a random pair of them with one heuristic (Gigerenzer, Todd, & Group, 1999). For ACE, a recognized choice is one made earlier in search (and subsequently retracted) on the same problem. On problems where
p. 55 retractions ↳ are common, reusing prior decisions with the highest weighted Advisor to break ties accelerated decision time, despite increased errors (Epstein & Ligorio, 2004).

Conclusions

On difficult problems, errors in a model of expertise may be inevitable, and training instances 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.

Constraint solving is a paradigm for many kinds of AI problems. ACE learns to solve problems in many difficult classes, where problems stymie off-the-shelf solvers that cannot monitor and modify their own behavior. 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.

Metareasoning is essential in FORR's ability to learn to solve problems within a given class. As it learns to manage heuristics, FORR uses metareasoning to decide when to abandon an unpromising learning attempt, when to stop learning, how to select heuristics during learning, and how to prioritize heuristics. FORR also reasons about its performance on previous problems, its previous decisions, and the relative discriminatory power of its heuristics. Thinking about thinking makes FORR-based applications more incisive and more successful.

Acknowledgments

This work was supported in part by the National Science Foundation under IIS-0811437 and IIS-0739122. ACE is an ongoing joint project with Eugene Freuder and Richard Wallace of The Cork Constraint Computation Centre.

References

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

Biswas, G., Goldman, S., Fisher, D., Bhuva, B., & Glewwe, G. (1995). Assessing design activity in complex CMOS circuit design. In P. Nichols, S. Chipman, & R. Brennan (Eds.), *Cognitively diagnostic assessment* (pp. 167–188). Hillsdale, NJ: Lawrence Erlbaum.

Borrett, J. E., Tsang, E. P. K., &and Walsh, N. R. (1996). Adaptive constraint satisfaction: The quickest first principle. In W. Wahlster
p. 56 (Ed.), *Proceedings of the 12th European Conference on Artificial Intelligence* (pp. 160–164). Chichester: Wiley. ↳

Crowley, K., & Siegler, R. S. (1993). Flexible strategy use in young children's tic-tac-toe. *Cognitive Science,* 17(4), 531–561. 10.1207/s15516709cog1704_3
Google Scholar WorldCat Crossref

D'Andrade, R. G. (1991). Culturally based reasoning. In A. Gellatly & D. Rogers (Eds.), *Cognition and social worlds* (pp. 795–830). Oxford: Clarendon.

Epstein, S. L. (1994a). For the right reasons: The FORR architecture for learning in a skill domain. *Cognitive Science,* 18(3), 479–511. 10.1207/s15516709cog1803_4
Google Scholar WorldCat Crossref

Epstein, S. L. (1994b). Toward an ideal trainer. *Machine Learning,* 15(3), 251–277. 10.1007/BF00993346
Google Scholar WorldCat Crossref

Epstein, S. L. (1995). Learning in the right places. *Journal of the Learning Sciences,* 4(3), 281–319. 10.1207/s15327809jls0403_2
Google Scholar WorldCat Crossref

Epstein, S. L. (1998). Pragmatic navigation: Reactivity, heuristics, and search. *Artificial Intelligence,* 100(1–2), 275–322. 10.1016/S0004-3702(97)00083-0
Google Scholar WorldCat Crossref

Epstein, S. L. (2001). Learning to play expertly: A tutorial on Hoyle. In J. Fürnkranz & M. Kubat (Eds.), *Machines that learn to play games* (pp. 153–178). Huntington, NY: Nova Science.

Epstein, S. L., Freuder, E. C., & Wallace, R. J. (2005). Learning to support constraint programmers. *Computational Intelligence,* 21(4), 337–371. 10.1111/j.1467-8640.2005.00277.x
Google Scholar WorldCat Crossref

Epstein, S. L., & Ligorio, T. (2004). Fast and frugal reasoning enhances a solver for really hard problems. In K. Forbus, D. Gentner & T. Regier (Eds.), *Proceedings of the 26th Annual Conference of the Cognitive Science Society* (pp. 351–356). Mahwah, NJ: Lawrence Erlbaum.

Fukunaga, A. S. (2002). Automated discovery of composite SAT variable-selection heuristics. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence* (pp. 641–648). Menlo Park, CA: AAAI Press.
Google Scholar Google Preview WorldCat COPAC

Gagliolo, M., & Schmidhuber, J. (2006). Dynamic algorithm portfolios. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*. Fort Lauderdale, FL. Available at http://anytime.cs.umass.edu/aimath06.
WorldCat

Gent, I. P., Prosser, P., Walsh, T. (1996). The constrainedness of search. *AAAI/IAAI, 1*, 246–252.
Google Scholar WorldCat

Gigerenzer, G., Todd, P. M., & Group, A. R. (1999). *Simple heuristics that make us smart*. New York: Oxford University Press.
Google Scholar Google Preview WorldCat COPAC

Gomes, C. P., & Sellman, M. (2004). Streamlined constraint reasoning. In M. Wallace (Ed.), *Principles and practice of constraint programming CP-2004* (vol. LNCS 3258, pp. 274–287). Berlin: Springer Verlag. 10.1007/b100482
Crossref

Gomes, C. P., & Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence,* 126(1–2), 43–62. 10.1016/S0004-3702(00)00081-3
Google Scholar WorldCat Crossref

Gomes, C. P., Selman, B., Crato, N., & Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning,* 24, 67–100. 10.1023/A:1006314320276
Google Scholar WorldCat Crossref

Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). Optimization by simulated annealing: An experimental
p. 57 evaluation;Part I, Graph Partitioning. *Operations Research,* 37, 865–892. ↳ 10.1287/opre.37.6.865
Google Scholar WorldCat Crossref

Keim, G. A., Shazeer, N. M., Littman, M. L., Agarwal, S., Cheves, C. M., Fitzgerald, J., et al. (1999). PROVERB: The probabilistic cruciverbalist. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence* (pp. 710–717). Menlo Park, CA: AAAI Press.
Google Scholar Google Preview WorldCat COPAC

Minton, S., Allen, J. A., Wolfe, S., & Philpot, A. (1994). An overview of learning in the Multi-TAC system. In M. L. Ginsberg (Ed.), *Proceedings of the First International Joint Workshop on Artificial Intelligence and Operations Research*. Technical report AFOSR-96-0394.

Nareyek, A. (2003). Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende & J. P. deSousa (Eds.), *Metaheuristics: Computer Decision-Making* (pp. 523–544). Boston: Kluwer.

Petrovic, S., & Epstein, S. L. (2006a). Full restart speeds learning. In *Proceedings of the 19th International FLAIRS Conference (FLAIRS-06)* (pp. 104–109). Menlo Park, CA: AAAI Press.
Google Scholar Google Preview WorldCat COPAC

Petrovic, S., & Epstein, S. L. (2006b). Relative support weight learning for constraint solving. In W. Ruml & F. Hutter (Eds.), *Proceedings of the Workshop on Learning for Search at AAAI-06* (pp. 115–122). Menlo Park, CA: AAAI Press.

Petrovic, S., & Epstein, S. L. (2008). Random subsets support learning a mixture of heuristics. *International Journal of Artificial Intelligence Tools,* 17(3), 501–520. 10.1142/S0218213008004023
Google Scholar WorldCat Crossref

Ratterman, M. J., & Epstein, S. L. (1995). Skilled like a person: A comparison of human and computer game playing. In J. D. Moore & J. F. Lehman (Eds.), *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 709–714). Hillsdale, NJ: Lawrence Erlbaum.

Schraagen, J. M. (1993). How experts solve a novel problem in experimental design. *Cognitive Science,* 17(2), 285–309. 10.1207/s15516709cog1702_4
Google Scholar WorldCat Crossref

Streeter, M., Golovin, D., & Smith, S. F. (2007). Combining multiple heuristics online. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)* (pp. 1197–1203). Menlo Park, CA: AAAI Press.
Google Scholar Google Preview WorldCat COPAC

Williams, R., Gomes, C., & Selman, B. (2003). On the connections between backdoors, restarts, and heavy-tails in combinatorial
p. 58 search. Paper presented at the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT03). ↳