Optimistic Problem Solving

Susan L. Epstein

Computer Science Department, Hunter College and The Graduate Center of The City University of New York
695 Park Avenue, New York, NY 10065, USA
susan.epstein@hunter.cuny.edu

Abstract

People are optimistic about problem solving. This paper identifies hallmarks of optimistic human problem solving and how people control for the errors it often engenders. It describes an architecture that permits a controlled version of optimistic problem solving and recounts experiments with it in three very different domains. Results indicate that, controlled for error, optimistic problem solving is a robust approach to a variety of challenging problems.

Introduction

Young children regularly tackle new problems expecting to succeed; they reuse decisions, rely on the familiar, and jump to conclusions. Such *optimistic problem solving* is quite different from traditional artificial intelligence programs with their insistence on proof, many trials, and extensive computation. This paper characterizes and analyzes optimistic problem solving, and offers examples of its impact in three very different domains. Particularly noteworthy are recent successes on difficult constraint satisfaction problems.

People make decisions and solve problems optimistically because it is *ecologically rational*, that is, computationally reasonable given their environment (Gigerenzer and Goldstein, 2002). The novelty, unpredictability, and breadth of experience inherent in the real world, and the need to respond to it in real time, demand more efficiency and less rigor. As computers tackle increasingly realistic tasks, ecological rationality becomes an important issue.

In contrast, global state-space search (hereafter, *search*) is a classical AI paradigm for problem solving. It moves from one *state* (description of the world) to another by selecting an action. Such search begins at an initial state and halts when it arrives at a state that satisfies some set of goal criteria. Unless it halts at a goal, a *complete* search method guarantees to visit every state. In a space of manageable size, complete search is tractable and foolproof, but it was developed for static, deterministic domains (e.g., chess). In more ambitious domains, complete search is often prohibitively expensive.

The next two sections of this paper assemble further observations about how people solve problems and control for error. Subsequent sections describe how one architecture capitalizes on these ideas, and recounts results in three domains: game playing, path finding, and constraint satisfaction.

Hallmarks of Optimism

Human problem solving has several hallmarks that distinguish it from search.

People expect to succeed. At the end of the day, most urban workers head home without checking the news to see if a power failure or a massive tidal wave has disabled mass transit. Assuming the unexceptional simplifies decision making. A rule like "To get home, take the subway" is almost always right. In contrast, some programs prove that an intended approach will solve the problem before executing the steps that comprise it. That would entail not only verifying that the subway is running, but that it will continue to do so and that particular trains will make all their scheduled stops, a computationally infeasible task. Heuristics are the AI version of people's optimistic rules, and the foundation of this paper.

People prefer to think less. Together with skill, computational speed is prized by people as a form of expertise (D'Andrade, 1991). A generally reliable rule like "Take the subway" conserves resources on almost every occasion. In contrast, search works hard to assure that its solution is correct: it looks at all the alternatives and plans for many contingencies. Two automated, somewhat primitive forms of thinking less are *bounded rationality*, which limits the resources available during computation, and *anytime* algorithms, which are interruptible but expected to deliver increasingly better solutions across time (Zilberstein, 1996).

People store and reuse old decisions. Once people learn a successful approach to a familiar problem, they reuse it. Only if traveling conditions change are people likely to reconsider "Take the subway." Reuse extends to portions of a solution as well. Expert chess players, for example, reuse old openings and agree on previously computed endgame outcomes to skip the final moves. Human experts under time constraints do *situation-based reasoning*, where procedures trigger in a familiar context, and produce a set of decisions (Klein and Calderwood, 1991). Placed in a

novel transportation facility, for example, our experienced subway rider will try to pay a fare, without checking to see if transport is free. AI has reused old decisions in case-based reasoning, in memoization, and in large knowledge bases (Hsu, 2002; Schaeffer et al., 2005), but rarely to the extent that people do.

People rely on familiarity when making decisions. There is an extensive literature on human preference for familiar choices, even when the context of a decision is different (Gigerenzer and Goldstein, 2002; Goldstein and Gigerenzer, 2002). *Fast and frugal reasoning* is an extensively studied, somewhat extreme example of human reliance on the familiar (Gigerenzer and Goldstein, 1996). It has been documented in people, particularly under time pressure. Fast and frugal reasoning can be paraphrased as "I once encountered this choice, reasoned about it (possibly in a different context), and now choose it once again." This is not case-based reasoning, because the only context is the choice set. It is as if one were choosing a mode of transportation in a foreign city: "They have a subway and I take one at home so I will do it here." High-level pseudocode for fast and frugal reasoning appears in Figure 1. Given a set of choices and a set of heuristics that return boolean values, it uses only one heuristic at a time to select one choice from among many. The basic heuristic for fast and frugal reasoning is simple recognition — it prefers the familiar. If more than one choice is recognized, a pair of recognized choices is selected at random. Next, a heuristic is selected and used to compare the two. If the heuristic prefers one of the pair to the other, the preferred choice is returned; otherwise another heuristic is chosen and the comparison is repeated on the same pair. Strategies people use to select a heuristic include random choice, the heuristic that most recently succeeded in breaking a tie, or the heuristic known to work best on the current problem set (*Take the Best*). In contrast, search typically treats the decision at each node as an entirely new computation.

People focus their attention. Chess grandmasters and Go masters visually attend to very few moves before they choose one, far fewer than weaker players (Holding, 1985; Kojima and Yoshikawa, 1998). In contrast, complete search uses ordering heuristics that sequence choices based on some metric, but it does not eliminate them from consideration.

```
Fast-and-frugal(choices, heuristics)
familiar ← Recognized(choices)
If |familiar| = 1
then return familiar
else
  chosen ← ∅
  pair ← randomly choose 2 from familiar
  Do until |chosen| = 1
    heuristic ← Strategy(heuristics)
    chosen ← prefer(heuristic, pair)
  return chosen
```

Figure 1: *Pseudocode for fast and frugal reasoning. Given a set of choices and a set of heuristics, this algorithm selects a single choice. The strategy for heuristic selection is key.*

People combine multiple heuristics to make decisions. While designing circuits or playing games, people use several heuristics simultaneously (Biswas et al., 1995; Ratterman and Epstein, 1995). In contrast, search that relies on multiple heuristics typically prioritizes them (Minton et al., 1995; Nareyek, 2003), reserves the more costly for harder problems (Borrett, Tsang and Walsh, 1996), or regards its heuristics as a portfolio of procedures that compete or alternate on the same problem (Gagliolo and Schmidhuber, 2007; Gomes and Selman, 2001; Streeter, Golovin and Smith, 2007).

Controlling for Error

Optimistic problem solving runs the risk of error. In particular, reuse of past computation is unsafe, because the current state of the world may not be sufficiently similar to the state in which that decision was originally formulated. For example, recent flooding or brownouts make the subway a less attractive alternative.

Optimism is different from naiveté, and people want to succeed. To control for the errors that their own optimistic problem solving engenders, people have important compensatory behaviors:

People evaluate their own behavior and adjust it accordingly. Human self-awareness includes perpetual performance evaluation. This evaluation is categorized by task similarity, that is, transportation decisions are different from chess moves. Self-judged poor performance is a strong motivation for behavioral change. Adaptive programs judge and change their own behavior too.

People evaluate their knowledge sources for trustworthiness. Even three and four year-old children monitor the accuracy of information from individual sources and use it to guide subsequent learning (Birch, Vauthier and Bloom, 2008). Despite their teachers' instructions, elementary school children often develop their own peculiar "rules" for how to subtract multi-digit numbers (Siegler and Crowley, 1994). When they do so, they incorporate only one such rule at a time into their existing algorithm, and use it determinedly until corrected. In contrast, programs often merely accept their heuristics at face value.

People learn from their mistakes. In the rainy season, getting wet every day soon loses its humor — people carry an umbrella. Although people may regard a few repetitions of the same error as humorous, they quickly label many repeated errors "unintelligent." People expect to learn how to avoid a particular repeated error. In contrast, search simply backtracks out of its mistakes.

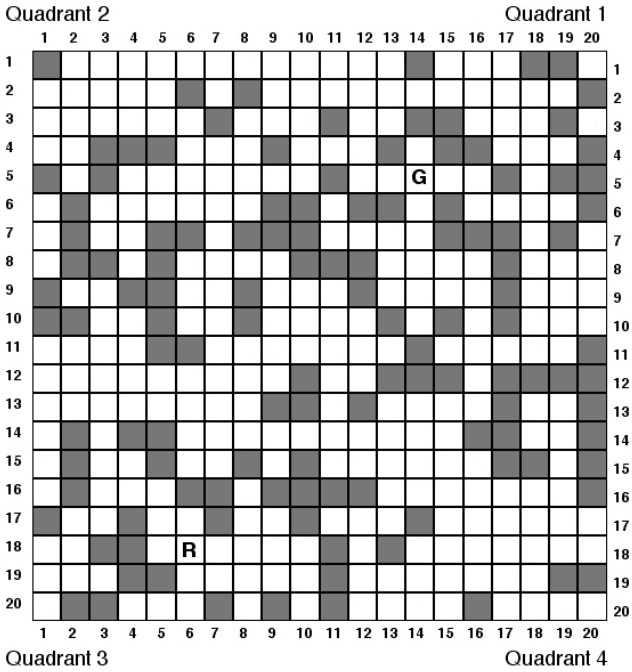

Figure 2: *A maze problem for Ariadne, where the robot R must reach the goal G. Each decision chooses a location on a vertical or horizontal line from the robot to the nearest (shaded) walls. The robot in this state has 8 possible moves and will require 12 to reach the goal.*

People integrate perception with reasoning. People bring to bear a full range of input modalities when problem solving, including vision, hearing, and smell. For example, people consider prowess at blindfolded chess remarkable because they believe vision is integral to a process that "sees" future game positions and examines them for flaws. Search represents some descriptive version of perception within the state description.

From Optimism to Architecture

We explore optimistic reasoning here with *FORR* (FOr the Right Reasons), an architecture for learning and problem solving (Epstein, 1992). FORR serves as a system shell. To produce a FORR-based program for a domain, one specifies a set of decision procedures (*Advisors*), a set of problems, and some knowledge to acquire.

We reference three FORR-based programs here, each of which learns to perform expertly in its domain. *Hoyle* is a game player with 19 different two-dimensional, finite-board games. *Ariadne* is a simulated robot pathfinder for two-dimensional mazes like that in Figure 2. *ACE* (the Adaptive Constraint Engine) solves binary constraint satisfaction problems (*CSPs)*. A binary CSP is a set of variables, each with a domain of values it can assume, and a set of constraints that restrict the way pairs of the variables can hold values simultaneously. Many CSP heuristics are based on the *constraint graph*, which represents variables as nodes and constraints as edges between them.

To solve a problem in its domain, a FORR-based program repeatedly makes decisions (e.g., moves in Hoyle or Ariadne, the selection of a variable or a value for it in ACE). Each decision incorporates advice from many Advisors. An Advisor either comments on individual choices or does situation-based reasoning. Other than input and output, no uniformity is imposed on Advisors; they are simply procedures. Every Advisor accepts as input the current state, the choices available to it, and any knowledge learned on previous problems. Every Advisor outputs a set of comments whose *strengths* indicate its degree of support for or opposition to some subset of those choices. Hoyle and Ariadne each have dozens of Advisors. ACE has more than 100 Advisors gleaned from the CSP literature; each of them either selects the next variable to be assigned a value or chooses a value for it.

To manage all that advice, FORR organizes decision making as a three-tier hierarchy where the first tier to make a decision does so, as shown in Figure 3. FORR distinguishes between Advisors that are correct and Advisors that are merely heuristic. Those Advisors that comment on individual choices and that the programmer judges to be both correct and quick are placed in tier 1. Tier 1 Advisors have the authority to make a decision immediately, or to eliminate some choices from further consideration. For example, Hoyle's *Victory* selects the final winning move in a game, Ariadne's *NoWay* vetoes moves that would re-enter a previously visited dead-end that does not contain the goal, and ACE's *Degree-Zero* opposes the selection of a

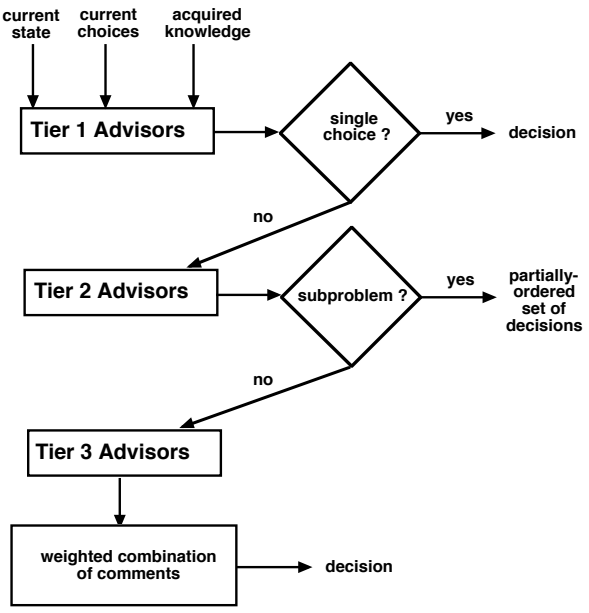

Figure 3: *FORR's 3-tier architecture of Advisors.*

$$\underset{j}{argmax}\left\{\sum_i d_i\, w_i\, c_{ij}\right\}$$

where
$g_i =$ number of opinions Advisor i has generated
$d_i = \begin{cases} 0.1 * g_i \text{ if } g_i < 10 \\ 1 \text{ otherwise} \end{cases}$
$w_i =$ weight of Advisor i
$c_{ij} =$ *strength* of comment from Advisor i on choice j

Figure 4: *Voting in tier 3. Comment strengths are normalized into a uniform interval; Advisor weights are learned.*

variable that has no neighbors in the constraint graph until all those with neighbors have been assigned mutually consistent values.

Advisors in tier 2 are triggered by familiarity and quickly produce a plan to address a recognized situation. For example, when Ariadne's robot is aligned with the goal but a wall lies between them, *Roundabout* generates a path to go around it. As a second example, ACE's *Separate* selects variables from a cyclic connected component in the constraint graph until the component becomes acyclic. The Advisors in tier 2 are also pre-sequenced by the programmer. The first plan they produce is accepted. *Executor*, a tier-1Advisor, monitors an executing plan's progress and has the ability to abandon it. Situation-based reasoning is not case-based; it deliberately avoids the subtleties inherent in indexing and partial matching — if there is no obviously recognized situation, FORR proceeds instead to tier 3.

The vast majority of Advisors in a FORR-based program comment on individual choices but are costly to compute or are not perfectly reliable. They are assigned to tier 3. Each tier-3 Advisor assigns a score to choices based on its underlying metric. For example, Ariadne's *Giant Step* prefers longer distances, and ACE's *Max Degree* prefers variables with a higher degree in the constraint graph. Tier-3 Advisors are all given the opportunity to comment on the choices remaining after tier 1 has filtered them and tier 2 has not proffered a plan. Typically, most decisions are made in tier 3, where *voting* selects the choice with the greatest support as the decision. A typical experiment with ACE, for example, made 87.1% of the variable decisions, and 98.9% of the value decisions in tier 3. Voting in FORR uses a weighted combination of comment strengths to calculate support. The computation for voting is shown in Figure 4.

FORR's modularity makes it an ideal test-bed for problem solving. It can use only a single heuristic, or a preordered set placed in tier 1. It will try to plan only if there are Advisors in tier 2. It combines heuristics in tier 3, where weights can be pre-specified or learned, and the weight-learning algorithm can be domain-specific. It can break ties in tier 3 lexically or at random, so that experiments are readily reproduced. For example, ACE generated the data in Table 1, which shows the performance of a variable-ordering heuristic and two value-ordering heuristics for CSP search. Although each heuristic solved every problem, the first pair (line 4) searched far fewer nodes and did so faster. The second pair (line 5) searched about as many nodes as the first pair, but took more than twice as long as Max Degree alone. This is an important issue in the use of multiple heuristics: more knowledge may make fewer mistakes (here, require backtracking out of fewer nodes), but it may also slow problem solving. Table 2 shows performance results on structured CSPs, which are more similar to those based on real-world problems. It demonstrates that different heuristics, and different combinations of heuristics, are more appropriate for different problem classes. Clearly, learning is essential.

Table 1: *A pair of heuristics outperforms individual ones on 50 CSPs with 30 variables and domain size 8, each of which has a potential search space of 9^{30} nodes. Problems are at the phase transition, that is, particularly hard for their size. Time is in CPU seconds; space is number of nodes in the search tree. Max Degree is a variable-ordering heuristic; Max Product and Promise are value ordering-heuristics (Geelen, 1992).*

Heuristics	*Space*	*Time*
Max Degree	139.70	0.435
Promise	770.14	3.037
Max Product	774.80	6.454
Max Degree + Promise	93.80	0.377
Max Degree + Max Product	99.28	0.836

Optimism Implemented

FORR's design is intended to support optimistic problem solving, which develops expertise quite rapidly in all three

Table 2: *Performance of heuristics on two classes of challenging, structured CSPs. (The geometric problems each have a search space of size 11^{50}, the composed problems of size 7^{30}.) Search was limited to 5000 nodes on each problem. ddd is a variable ordering metric that calculates the ratio of the number of values that a variable could be assigned in the current state to the number of unassigned neighbors it has in the constraint graph. v_1 and v_2 are value-ordering heuristics. After inference, v_1 orders values by the smallest domain size produced, and v_2 orders values by the largest product of all domain sizes. Values are averaged over 50 problems. Space is nodes searched; elapsed time is in CPU seconds. (Reproduced from (Epstein and Petrovic, 2008).)*

	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%

domains. Ten trips in a maze, 30 CSPs of similar size and difficulty, or 50 – 100 contests at a game are generally enough to produce outstanding performance (Epstein, 1995; Epstein, 2001; Epstein, Freuder and Wallace, 2005).

Many of the hallmarks of optimistic problem solving cited earlier are addressed inherently, by FORR's design:

• Expectation of success is embodied in extensive reliance on tier-3 heuristics and on tier-2 situation-based reasoning.

• Thinking less motivated the placement of faster procedures in tier 1 and acceptance of the first plan created. Situation-based reasoning is the foundation of tier 2, where reused procedures address subgoals.

• Reliance on the familiar (fast and frugal reasoning under a particular strategy for heuristic selection) can be implemented by a single Advisor in tier 1.

• Focus of attention motivated the ability to veto choices in tier 1.

• Reliance on many knowledge sources is implemented as multiple Advisors.

Learning is an integral part of the architecture. FORR learns on a set of problems characterized as similar (e.g., contests at the same game or trips between different locations in the same maze). FORR evaluates its performance after every problem and learns from it. For example, tier-3 Advisors are provided to a FORR-based program because they are expected to be accurate in some situations, on some problem classes. Which Advisors are appropriate, and to what extent, is learned as the weights w_i in Figure 4. Domain-specific knowledge, such as dead-ends in a maze or forks in a game, may also be learned.

An *experiment* in FORR averages behavior over a set of runs. Each *run* is a self-supervised learning phase followed by a testing phase. The acquired knowledge (e.g., Advisor weights) is then made available during search on subsequent problems. During testing, learning is turned off, but the acquired knowledge remains available to the Advisors. The last line in Table 2 demonstrates how well ACE can learn from only 30 CSPs given 40 Advisors. Only ACE's mixture solves every problem within the space limit.

Like people, a FORR-based program takes precautions against the errors its optimism may engender. Weight learning, for example, evaluates knowledge sources for trustworthiness. It extracts examples of both good and bad decisions from the trace of each successful problem solving experience, and uses them as training examples. Weights are reinforced based in part on the difficulty of each decision and the magnitude of each error. (For further details on reinforcement weight learning in FORR, see (Epstein, 1994; Epstein and Petrovic, 2008).) During voting, the discount factor d_i in Figure 4 gradually introduces Advisors into the mix. It modulates their influence until they have commented often enough for weight learning to evaluate their trustworthiness accurately, much the way people tentatively integrate new rules.

The Impact of Optimism

This section recounts experiments that further emphasize optimism in human-like ways.

Learning How to Solicit Less Advice

Because most Advisors reside in tier 3, and because many of their metrics rely on costly state representations, tier-3 Advisors consume the vast majority of a FORR-based program's computing time. Thinking less could therefore be interpreted as consulting fewer of them for advice. This conflicts, however, with the tenet that multiple heuristics are better. Nonetheless, it is possible to reduce the number of Advisors and the frequency with which they are consulted to accelerate problem solving without impairing performance.

The learning phase in an experiment identifies trustworthy knowledge sources appropriate to a particular problem class: those tier-3 Advisors with the highest weights. Because people seek at least better-than-random performance, FORR provides for *benchmark* Advisors that make randomly many comments with random strengths. (Ariadne and Hoyle have a single benchmark Advisor; ACE has one for variable selection and one for value selection.) Benchmark Advisors do not participate in search decisions, but they do receive weights during the learning phase. Then, during testing, only Advisors whose weights are greater than that of their respective benchmarks are consulted. Because some Advisors' costly-to-compute representations are shared, however, speedup is not always equivalent. Typically, about half of ACE's Advisors are eliminated by the benchmarks, and this accelerates testing by about 30%.

One might think that an exceptionally highly weighted tier-3 Advisor ought simply to be *promoted* to tier 1 during testing. Experiments with Hoyle, however, indicate that even an Advisor that has been 99.5% correct during learning should not be promoted from tier 3 to tier 1 (thereby bypassing tiers 2 and 3 for most decisions) because a single error in play can prove fatal. There are more subtle and successful ways to exploit learned weights.

For example, many CSPs have a *backdoor*, a relatively small set of variables that, once assigned the right values, make search nearly trivial (Williams, Gomes and Selman, 2003). Once past the backdoor, it should be safer to think less. Although identification of the backdoor in a particular problem is NP-complete, ACE can conservatively (over)estimate the size of the backdoor b on a class of CSPs as the point after which there was no backtracking in any solved problem during the learning phase. *Pusher* is a tier-1 Advisor, active only during testing, that, after b variables have been valued, consults only the single highest-weighted tier-3 Advisor to force a decision. Unless Pusher's Advisor is wrong, this should accelerate search. We tried three versions of Pusher: one pushed all deci-

sions, one pushed only for value selection, and one pushed only for variable selection. The variable-only version was the best. In experiments on a broad variety of problem classes, it never impaired search performance, and it usually accelerated testing by about 8%. After a while, it is apparently safe to think less about which part of the problem to address next, but not about what to do there. Pushing is an effective way to jump to a conclusion because it has a context, "after the backdoor," and a breadth of experience behind it. An accurate estimate for b is essential, however. Pushing too soon (overly low b) can prove as problematic as promotion, and pushing too late (overly high b) can produce little performance improvement.

Prioritization is a more tentative way to push. It seeks to exploit top-weighted tier-3 Advisors first, and resorts to the remainder only to break ties. Prioritization partitions tier 3 into k subsets, S_1, S_2, …, S_k. Subset S_1 votes first. If it can make a choice it does so, otherwise only the tied, top-rated choices are forwarded to S_2, which then votes, and so on. In this way, fewer Advisors are likely to be consulted on fewer choices. The issue, however, is how to partition the Advisors after they have been filtered by their benchmark. In extensive testing, we have found that it is more effective to cluster Advisors according to weight intervals of equal size than to form subsets of equal size. Given a set of 40 Advisors, ACE does best with this approach when it identifies 3 subsets. Prioritization accelerates testing performance by about 5%.

Ranking is a more extreme version of prioritization; it replaces voting in tier 3 with a list of Advisors ordered by weight. This is equivalent to prioritization where all subsets are of size 1. Ranking has never proved better than prioritization in any of these domains, and is often worse.

Reusing Decisions

The impact of fast and frugal reasoning has thus far been explored only in ACE. As implemented here, familiarity requires errors — after backtracking, a variable that no longer has an assigned value and the retracted value for it will both be familiar and preferred when they recur later among the current choices. The premise is that variables and values for them that were once attractive to the heuristics will still be attractive, so that re-consulting the Advisors that selected them is unnecessary. Certainly, the classical context in which they were selected has changed, because the kind of constraint search described here never revisits exactly the same CSP node. Nonetheless, fast and frugal reasoning optimistically assumes that there is not enough contextual difference to re-consult all of tier 3. If multiple choices are recognized, fast and frugal reasoning selects a pair of recognized choices at random and uses one Advisor at a time until some Advisor prefers one choice to the other. The algorithm in Figure 1 was applied to the problems in Table 1 only during the testing phase. With the *Take the Best* strategy for heuristic selection (in descending order of weight), fast and frugal reasoning reduced ACE's computation time by 24%, despite the introduction of 8% more errors (Epstein and Ligorio, 2004). This approach requires errors to generate familiarity, however, so performance improves less on easier problem classes. At the other extreme, if problems are very hard and search is extensive, most variables and values for them may be familiar. In that case, fast and frugal reasoning could degenerate into ranking, which we know to be inferior to a weighted mixture.

Another successful but startlingly optimistic reuse of prior decisions is evident in the reuse of locations once visited by Ariadne's robot (Epstein, 1998). After each trip, Ariadne analyzes the trace, and removes unnecessary digressions. The locations remaining may have facilitated travel because they provided access to a different quadrant of the maze (*gates*), offered the possibility of a new direction (*corners*), or merely represented a counterintuitive step that did not direct the robot toward the goal (*bases*). These *facilitators* are pragmatic descriptions of the maze. Some tier-3 Advisors (e.g., *HomeRun*) advise the robot to move toward them; some tier-2 Advisors (e.g., *Patchwork*) use them to formulate plans quickly and optimistically, assuming that an obstruction will not intervene. (Such partially-executed plans are discarded as soon as they fail.) Because bases are reinforced each time they are revisited, even if a plan including them fails, Ariadne develops habitual behaviors in the same maze, much like "Take the subway."

A final example of decision reuse is Hoyle's learned move sequences (Lock and Epstein, 2004). These sequences are valued according to the outcome of play (win, loss, or draw). Unlike fast and frugal reasoning, each move sequence has its own context, a generalized, pattern-like description of the location of pieces on the board immediately before each time the sequence was put into effect. Hoyle uses these sequences in situation-based reasoning: when the current board matches the context of a highly valued sequence, Hoyle follows its advice. For five men's morris, a game with millions of nodes in its search space, Hoyle learned sequences while it observed 40 contests between programs at various skill levels. Reusing this sequence knowledge, and with no other Advisors, Hoyle was able to draw more than half the time against an external program that played flawlessly, and to perform well against the three other programs it had observed.

Discussion

There are, of course, any number of domains in which optimistic reasoning would be inappropriate. The domain must have a high tolerance for error; nuclear power generation would not qualify. It must also have a low recovery

cost; although backtracking is inexpensive, it may be forbidden, as in game playing. The domain must also be consistent enough so that reuse is reasonable.

There are also requirements for learning. The environment must be extensive enough and broad enough to prepare the learner for a variety of situations, the way Hoyle's sequence learning environment did. The learner must be self-aware and reward efficiency. There must be mechanisms that compactly capture and flexibly reuse regularities from experience, such as Ariadne's facilitators. There must be some standard for performance, gauged if need be by the learner's best. There must be a mechanism to detect and learn from errors as well as successes.

The integration of perception with reasoning often leads to dramatic performance improvements. Hoyle's learned move sequences are one example. Ariadne contains many others; obstructions, corridors, and dead-ends all summarize the robot's experience as it senses its way through a maze. Moreover, many CSP heuristics are rooted in the constraint graph, a representation that people use to visualize the problem. In FORR, most representations of the current state are shared, and computed only when requested by an Advisor. As a FORR-based program consults fewer Advisors, it is likely to compute fewer representations, and thereby become even more efficient, attending only to what it needs.

FORR's modularity permits us to explore how various implementations of optimism impact different applications, and impact different kinds of problems within a single application. The user of a FORR-based program specifies the Advisors in each tier for an experiment. As a result it is possible to test individual Advisors and tiers, as well as any combination of them. Because promotion, prioritization, pushing, fast and frugal reasoning, and the various weight-learning algorithms are optional, one can experiment to gauge their impact. This paper summarizes thousands of experiments. Not surprisingly, the gentler forms of optimism (e.g., prioritization) are very successful on easy problems, and the more extreme forms (e.g., promotion) can considerably degrade performance on hard problems. Repeatedly, however, we have observed that some degree of optimism is surprisingly effective.

Still, optimistic problem solving, as documented in people, often strikes computer scientists as foolhardy. Approaches that entail random choice and lazy calculation do not support the rigor to which we are accustomed. Nonetheless, human problem solving is robust, efficient, and surprisingly effective. The results presented here suggest that optimism, controlled for error, merits our further consideration.

Acknowledgements

This work was supported by the National Science Foundation under 9222720, IRI-9703475, IIS-0328743, IIS-0739122, and IIS-0811437. ACE is an ongoing joint project with Eugene Freuder and Richard Wallace. Joanna Lesniak, Tiziana Ligorio, Xingjian Li, Esther Lock, Anton Morozov, Smiljana Petrovic, Barry Schiffman, and Zhijun Zhang have all made substantial contributions to this work.

References

Birch, S. A. J., S. A. Vauthier and P. Bloom 2008. Three- and four-year-olds spontaneously use others' past performance to guide their learning. *Cognition* 107(3): 1018-1034.

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 *Proceedings of 12th European Conference on AI*, 160-164.

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. 1992. Prior Knowledge Strengthens Learning to Control Search in Weak Theory Domains. *International Journal of Intelligent Systems* 7: 547-586.

Epstein, S. L. 1994. Identifying the Right Reasons: Learning to Filter Decision Makers. In *Proceedings of AAAI 1994 Fall Symposium on Relevance*., 68-71. New Orleans, AAAI.

Epstein, S. L. 1995. On Heuristic Reasoning, Reactivity, and Search. In *Proceedings of Fourteenth International Joint Conference on Artificial Intelligence*, 454-461. Montreal, Morgan Kaufmann.

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 *Proceedings of Cognitive Science 2004*, 351-356. Chicago, Lawrence Erlbaum.

Epstein, S. L. and S. Petrovic 2008. Learning Expertise with Bounded Rationality and Self-awareness. In *Proceedings of AAAI Workshop on Metareasoning*, Chicago, AAAI.

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

Geelen, P. A. 1992. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *Proceedings of 10th European Conference on Artificial Intelligence*, 31-35.

Gigerenzer, G. and D. G. Goldstein 2002. Models of Ecological Rationality: The Recognition Heuristic. *Psychological Review* 109(1): 75-90.

Gigerenzer, G. and G. Goldstein 1996. Reasoning the Fast and Frugal Way: Models of Bounded Rationality. *Psychological Review* 103(4): 650-669.

Goldstein, D. G. and G. Gigerenzer 2002. Models of ecological rationality: The recognition heuristic. *Psychological Review* 109: 75-90.

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

Holding, D. 1985. *The Psychology of Chess Skill*. Hillsdale, NJ, Lawrence Erlbaum.

Hsu, F.-h. 2002. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton, NJ, Princeton University Press.

Klein, G. S. and R. Calderwood 1991. Decision Models: Some Lessons from the Field. *IEEE Transactions on Systems, Man, and Cybernetics* 21(5): 1018-1026.

Kojima, T. and A. Yoshikawa 1998. A Two-Step Model of Pattern Acquisition: Application to Tsume-Go. In *Proceedings of First International Conference on Computers and Games*, 146-166.

Lock, E. and S. L. Epstein 2004. Learning and Applying Competitive Strategies. In *Proceedings of AAAI-04*, 354-359. San Jose.

Minton, S., J. A. Allen, S. Wolfe and A. Philpot 1995. An Overview of Learning in the Multi-TAC System. In *Proceedings 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.

Ratterman, M. J. and S. L. Epstein 1995. Skilled like a Person: A Comparison of Human and Computer Game Playing. In *Proceedings 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 *Proceedings of IJCAI-05*, 292-297.

Siegler, R. S. and K. Crowley 1994. Constraints on Learning in Nonprivileged Domains. *Cognitive Psychology* 27: 194-226.

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

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

Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *The AI magazine* 17: 73-83.