Fast and Frugal Reasoning Enhances a Solver for Hard Problems

Susan L. Epstein (susan.epstein@hunter.cuny.edu)
Tiziana Ligorio (t.ligorio@verizon.net)
Department of Computer Science,
Hunter College and The Graduate Center of The City University of New York
695 Park Avenue, New York, NY 10021 USA

Abstract

This paper describes how a program that learns to solve hard problems has been enhanced with fast and frugal, recognition-based reasoning methods. The program uses these methods to help manage its own heuristics for solving constraint satisfaction problems. The result is a constraint solver that reasons quickly, much the way people appear to induce reasonable decisions in real-world situations. Empirical evidence on a variety of problems indicates that fast and frugal reasoning can accelerate the solution of very difficult problems, often without introducing additional error.

The thesis of this work is that the same *fast and frugal* reasoning which people use to formulate decisions in real-world situations (Gigerenzer, Todd, & The ABC Research Group, 1999) can accelerate autonomous decision makers without endangering their reliability. We investigate this premise with a program that learns how to combine heuristics to solve constraint satisfaction problems (*CSPs*). The principal result reported here is that, on difficult CSPs, when recognition alone is not sufficient, fast and frugal, recognition-based reasoning enhances the solver. This is particularly noteworthy because the program first learns how to apply its CSP heuristics within its problem environment, and then hones its performance using fast and frugal reasoning. We believe this to be the first exploration of fast and frugal reasoning on a large body of challenging problems whose difficulty can be explicitly characterized and whose solution can be incisively assessed.

Many large-scale, real-world problems in areas such as design and configuration, planning and scheduling, and diagnosis and testing are readily understood, represented, and solved as CSPs. CSP solution is, in general, not known to be solvable by algorithms of any but exponential complexity. Thus the effectiveness of fast and frugal reasoning on these problems is counterintuitive.

Fast and frugal reasoning assumes pre-acquired, accurate, problem-area knowledge (Gigerenzer, Todd, & The ABC Research Group, 1999). In real-world decisions, fast and frugal reasoning adaptively exploits the environment's structure. A program provided with heuristics must learn their accuracy before turning to fast and frugal decision making. The first sections of this paper provide background information on fast and frugal reasoning, and on CSP. Subsequent sections describe how we addressed these ideas in a program that learns, describe the experimental design, discuss the results, and sketch future work.

Background

As often happens in interdisciplinary work, terminology overlaps here, but meaning does not. Thus we alert the reader to the fact that, although "domain" means "problem area" in some fields, it has a different connotation (described below) in constraint solving, for which we reserve it. Similarly, fast and frugal researchers generally refer to their general problem-solving methods (e.g., Minimalist, Take the Last, Take the Best) as heuristics, but so too do CSP researchers, and again we take the CSP definition. As a result, we describe fast and frugal methods as "strategies that consult heuristics" rather than "heuristics that consult cues." Finally, the notion of recognition, which underlies the fast and frugal strategies, is itself a heuristic, albeit a more general one, defined below.

Fast and frugal reasoning

Under limited time, there exists a trade-off between decision making speed and correctness. When pressed for time, people may limit their search for information to guide them in the decision process with *non-compensatory* strategies, strategies that use a single heuristic to prefer a single option (Gigerenzer & Goldstein, 1996). People appear to work from an *adaptive toolbox*, a collection of cognitive mechanisms for inference in specific problem areas (Gigerenzer, Todd, & The ABC Research Group, 1999). This adaptive toolbox includes low-order perceptual and memory processes, including fast and frugal strategies that may be combined to account for higher-level mental processes. Such a model of cognitive heuristics is *ecologically rational*, grounded in environment-specific structure and characteristics (Goldstein & Gigerenzer, 2002; Gigerenzer & Selten, 2001; Gigerenzer, Todd, & The ABC Research Group, 1999).

In *one-reason* decision making, a set of heuristics is consulted one at a time, until some heuristic is able to *discriminate*, that is, able to select a single option. *Recognition* is the *foundation heuristic* for fast and frugal reasoning: it favors recognized options over unrecognized ones. Recognition discriminates if and only if exactly one option is recognized (Gigerenzer & Goldstein, 1996). In that case, it is sufficient for one-reason decision making, and no further computation is required.

When recognition alone does not discriminate, each strategy considered here may be thought of as a meta-heuristic that speeds the selection of the next heuristic. All three strategies initially try recognition on all the available options. If more than one option is recognized, other heuristics are consulted, one at a time, on a randomly-selected pair of recognized options, until some heuristic does discriminate. The preferred option is then selected. The way these "other" heuristics are chosen defines the decision-making strategy. The following are drawn from Gigerenzer, Todd and the ABC Research Group (1999):

- *Minimalist*: Select a heuristic at random, until one discriminates among the options and a decision is made.
- *Take the Last*: Use the last heuristic, the one that discriminated the last time a decision was made, when recognition did not. This captures the human tendency to re-use the most recent successful strategy.
- *Take the Best*: Use the heuristic known to work best in a specific environment. The insightfulness of a heuristic on a set of problems is called its *ecological validity*.

Constraint satisfaction problems (CSPs)

CSPs are a good vehicle with which to explore fast and frugal reasoning. They arise in classes whose difficulty has a mathematical characterization, and many examples can be readily generated within each class. Furthermore, well-established criteria exist with which to gauge the performance of a program that solves them.

A CSP consists of a set of variables, each associated with a *domain* of possible values for assignment, and a set of *constraints* that specify which combinations of values are allowed. To represent a real-world problem as a CSP, one casts the entities involved as variables, and expresses the relationships required among these variables as constraints. A simple example appears in Figure 1. Real-world CSPs, however, involve many more variables, substantial domains, and a broad variety of interacting, more sophisticated restrictions as constraints. A solution for a CSP is one value for each variable such that all constraints are satisfied. Every CSP has an underlying *constraint graph* that represents each variable by a vertex. An edge in the graph represents a constraint between the corresponding variables, and is labeled by their permissible pairs of values. The *degree* of a variable is the number of edges to it in the graph. (For simplicity we restrict discussion here to binary CSPs.)

How hard a CSP is to solve is determined by how difficult it is to find values that satisfy all its constraints at once. A *class* of CSPs groups together problems with four parameters thought to estimate their difficulty. A CSP class may be described by <*n*,*k*,*d*,*t*>, where *n* is its number

Variables: A, B, C, D

Domains: A is 1 or 2
B is 1, 2, or 3
C is 1, 2, or 4
D is 1 or 3

Constraints: A = B
C < D
D – A is even

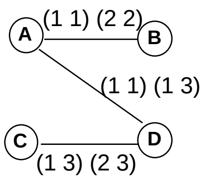

Figure 1: A simple constraint satisfaction problem and its underlying constraint graph. Edges in the graph are labeled with acceptable value pairs, computed from the domains and the constraints.

of variables and *k* its maximum domain size. The *density d* is the fraction of possible edges in the underlying constraint graph. The *tightness t* is the percentage of possible value pairs the constraints exclude. Thus in <30,8,.26,.66> every problem has 20 variables, each with domain size at most 8, and 30^8 possible value assignments. In a given class, every CSP has the same values for *n*, *k*, *d*, and *t*, and the same minimal number of decisions for solution.

To solve a CSP, one can repeatedly select a variable and assigns it a value consistent with its constraints. For example, Figure 2 represents a possible search for a solution to the problem in Figure 1. Reading from top to bottom and from left to right, each circle (*node*) represents a decision. There, Variable A was selected first, and assigned the value 2, then D was selected, and both its values (1 and 3) were tried without success. Therefore, search backed up, the value 2 was withdrawn (*retracted*) from A, and the value 1 was assigned to A instead.

The black nodes in Figure 2 represent the correct decisions, and the solid path on the right represents the solution. When a value assignment is inconsistent with the constraints, it is retracted and another assignment is tried. In Figure 2, white nodes are errors, assignments that cause subsequent decisions (the gray nodes) to be retracted, so that an error can be corrected. For example, the assignment D = 1 is *inconsistent* because it leaves no values for C. When that happens, values are retracted until all current assignments are once again consistent, and new values tried. Finding a single solution to a solvable problem this way requires at least *n* assignments.

Although CSP solution is NP-hard, some problem classes surrender readily to heuristics. For a solvable CSP, the order in which one selects variables (*variable ordering*, e.g., A, D, C, B in Figure 2) can speed solution, as can the order in which one assigns a value to a just-selected variable (*value ordering,* e.g., 2, 1 for A in Figure 2). There are dozens of variable-ordering and value-ordering heuristics in the CSP literature, but their interactions are ill-understood. The best approximation for CSP problem difficulty is currently *kappa*, which is defined as a function of *n, k, d,* and *t* (Gent, MacIntyre, Prosser, & Walsh, 1996). Nonetheless, two problems from the same class may still require different amounts of effort to solve.

During search, a CSP solver can apply a variety of inference and retraction methods. When a *partial solution* (a set of values assigned to a proper subset of the variables) is incompatible with the constraints, all the nodes that include it (e.g., gray in Figure 2) may be eliminated. An *inference* method can propagate the implications of a newly-assigned value on to the remainder of the as-yet-unassigned variables. Different amounts of inference are possible, and there are tradeoffs between inference effort and search savings. A specific inference method (the central one is arc consistency) can be carried out to varying degrees (Sabin & Freuder, 1994) and with different algorithms (Bessière & Régin, 2001). A *retraction* method responds to an inconsistent partial solution (a subtree of discarded nodes rooted at an error node in Figure 2). The standard retraction method is *chronological backtracking*, withdrawal of the most recent assignment(s).

ACE

ACE (the Adaptive Constraint Engine) is an autonomous system that learns to solve classes of CSPs (Epstein, Freuder, Wallace, Morozov, & Samuels, 2002). ACE is based on *FORR* (FOr the Right Reasons), a cognitively-oriented architecture for learning and problem solving that supports the development of expertise (Epstein, 1994). Here, "cognitively-oriented" means that FORR's reasoning structure emulates approaches readily observable in human problem solving, highly-effective approaches not always found in traditional AI artifacts (Biswas, Goldman, Fisher, Bhuva, & Glewwe, 1995; Crowley & Siegler, 1993; Klein & Calderwood, 1991; Kojima & Yoshikawa, 1998; Novick & Coté, 1992; Schraagen, 1993). FORR is based on the premise that decisions are composed from more and less trustworthy rationales. One constructs a FORR-based program by specifying heuristics that underlie decision making in a particular problem area.

ACE is an ambitious program – armed with many heuristics, it can tackle difficult problems. The version we used here begins with 40 CSP heuristics which are initially classified into a hierarchy of three *tiers* by the user. ACE moves through those tiers to make a decision. The heuristics in tiers 1 and 2 are consulted sequentially; the heuristics in tier 3 are consulted (effectively) in parallel.

Tier 1 consists of *perfect* (i.e., error-free) heuristics consulted sequentially. If any heuristic supports an action, that action is executed without reference to any subsequent heuristics. Consulting perfect heuristics first ensures that obvious correct decisions (e.g., a checkmate at chess) will be reached without devoting resources to other, less reliable heuristics. A perfect heuristic that opposes an action (e.g., "don't move into checkmate") removes that action from consideration by all subsequent heuristics, thereby preventing obvious errors. Thus placing perfect heuristics in tier 1 permits easy problems to be solved easily, a feature all too rare in complex systems. (The second tier is not applicable to the work reported here; the interested reader is referred to (Epstein, 1998)).

Tier-3 heuristics are the ordinary ones; they produce single-action *comments* that are not guaranteed to be correct. All but two of the ACE heuristics in this version lie in tier 3. Because they are fallible, their comments are combined to select the next action in a process called *voting*. Each heuristic may vote on different actions with different strengths, or it may remain silent. ACE's tier 3 heuristics were, for the most part, drawn from the CSP expert community. In every case, however, the dual of a popular heuristic was also implemented. For example, the CSP literature suggests that the next variable to have a value assigned to it should have a minimum *dynamic domain size* (set of values that would still be consistent with existing variable assignments). ACE therefore has a heuristic that comments in favor of such variables, but it also has its dual, a heuristic to maximize the dynamic domain size of a variable.

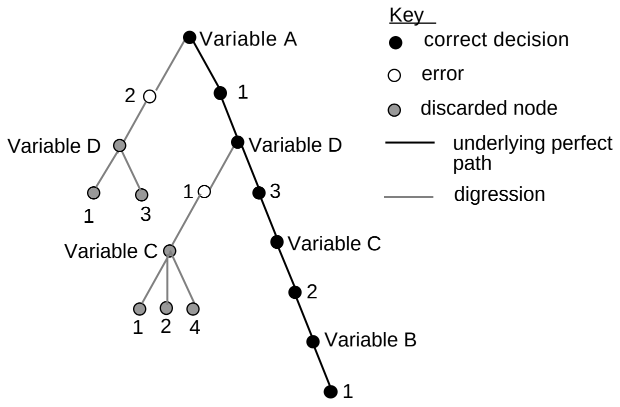

Figure 2: A search tree for solution to the simple CSP in Figure 1, depicted here as alternating variable selections and value selections. When a value selection violates some constraint, that decision is retracted, and search backtracks. New values are assigned until all variables have a value consistent with the constraints.

One-Reason Decision Making in ACE

ACE learns to solve CSPs efficiently by winnowing through its heuristics and balancing them appropriately. Laden with CSP knowledge, ACE's decisions are carefully reasoned but time-consuming. Fast and frugal, one-reason decision making seemed a reasonable enhancement, but its adaptation for ACE required careful thought about recognition, ecological rationality, ecological validity, and preference functions.

Recognition in ACE

Recognition is familiarity with something previously experienced. Recall that, during CSP solution, there are only two kinds of experience: variable selection and value selection. Therefore, we define recognition to be the identification of a current option as one previously selected during search *in the current problem*. Note that recognition for ACE is a selection heuristic rather than a trigger for situation assessment and re-evaluation, because mere recognition of a single option is sufficient for decision making without any consideration of the similarities of the different search states (situations) in which it previously occurred. For example, in Figure 2 assigning 1 to Variable D after Variable A is set to 1, eventually proves to be an error, since all possible assignments to Variable C are then incompatible with the constraints. In response to the detected error, the gray nodes corresponding to the selection of Variable C and the values tried for it are retracted. The decisions made in this subtree, *excluding* the error node, will be recognized during subsequent search. Effectively, later in search, a decision point that includes options previously found attractive will find them attractive again.

A counterargument to the role of recognition in human reasoning is that recognition is often associated with other cues to obtain a given judgment, and is thus *compensatory* (Oppenheimer, 2003). When an object is recognized, other information about that object is also recalled at the moment of recognition. If this "extra" information is relevant to the judgment being made, it will be taken into account while making the decision. The relevant information may support or oppose the judgment about the recognized object. Although the additional information may be contradictory, for our purposes, the influence it has on recognition is considered as a whole. Recognition may have an attached positive or negative correlation with respect to the judgment being made.

For ACE, recognition always has a positive correlation, because it treats a previous decision as one it wants to make again. The idea here is that, if no further knowledge about ACE's tier 3 heuristics has been acquired during search on the current problem, and consulting these same heuristics previously led to making certain decisions, they should remain valid, if the option is still available. (Consistency checking may have eliminated it.) By "recognizing" such previously-computed but subsequently-retracted decisions, ACE can avoid reconsulting all its heuristics on the same (or most of the same) options. For example, in Figure 2, ACE initially assigned 2 to A and then chose Variable D. Later, when 2 is retracted and 1 assigned to A, the next variable must be selected. At that point D is recognized and so need not be recomputed. Note too that recognition can lead ACE to repeat an error. It tries 1 before 3 for D on both sides of the search tree in Figure 2.

Ecological rationality and ecological validity

ACE must have acquired problem-class-specific knowledge before it attempts speed-up through one-reason decision making. Tier-3 heuristics are ACE's version of the adaptive toolbox, its knowledge about how CSP works. ACE's heuristics, however, are not all of equal significance or reliability in a particular problem class. Therefore, ACE learns weights to combine them.

DWL (Digression-based Weight Learning) learns problem-class-specific weights for tier-3 heuristics (Epstein et al., 2002). It is specifically designed to minimize errors during solution (and therefore minimize the number of nodes in the tree of Figure 2). After ACE solves a problem, DWL examines the solution trace, and adjusts the weight of each heuristic according to whether or not it supported the correct decisions. All heuristics start with the same weight. DWL provides the ecological validity for the heuristics in a given problem class.

DWL also employs non-voting, baseline heuristics that discriminate with randomly-generated strength on randomly-chosen options. DWL uses them to gauge ACE's own heuristics, so that ACE learns to value only those heuristics that make comments more valuable than random ones. These baseline heuristics provide ACE's ecological rationality.

From preference to binary decisions

The one-reason decision-making model assumes binary heuristics, ones that vote either in favor of or against an option. Recall that, if recognition does not discriminate, the model considers other heuristics on randomly-selected pairs of options, until some heuristic discriminates. Recognition is a binary heuristic, and we translate it as such for ACE: a decision was either previously made or not. A tier-3 heuristic, however, expresses its preference for, or opposition to, an option in a comment whose *strength* lies between 0 and 10. To adapt ACE for one-reason decision making, the option with the higher strength is deemed the positive one. If a heuristic comments on both options with the same strength, or it does not comment at all, the heuristic does not discriminate, and another heuristic is consulted, depending upon the particular strategy in use.

Experimental Design

The ACE project maintains a large library of problem classes, each with many examples. In each experiment here, ACE learned by attempting to solve at most 600 problems (the *learning phase*) and then was tested on 200 different problems from the same class, with learning turned off (the *testing phase*). This learn-and-then-test approach was repeated 10 times, each time with different learning problems but the same testing problems. Fast and frugal reasoning was applied only in the testing phase.

ACE's performance here is evaluated by three standard CSP criteria: average number of nodes in the decision tree (e.g. Figure 2), average number of mistakes during solution (number of retractions), and average computation time (in seconds). Any differences identified in the following discussion are statistically significant at the .95 level. Learning was terminated early if the heuristics' weights *stabilized* (did not vary in their standard deviation by more than 0.1 over the most recent 20 problems) before 600 problems. ACE used chronological backtracking for retraction and MAC3 (Mackworth, 1977) for consistency checking. What we varied in our experiments was the problem class, and which non-compensatory search strategy was used in the testing phase.

Results

We tested ACE alone, and then with each of the fast and frugal strategies on each of three problem classes. The results appear in Table 1. The first class of problems was <30, 8, .26, .66>, an extremely difficult set of randomly-generated CSPs. (The state of the CSP art does not yet support labeling them "the most difficult" for their size, but these are certainly "exceptionally difficult.") On this class, each of the three non-compensatory strategies, combined with recognition significantly improved overall execution time. Speed-up came with a price, however. Although ACE solved every problem, it made more (albeit relatively trivial) errors, and explored more nodes during search. The most reasoned and ecologically ra-

Table 1: Performance of ACE alone and with the recognition heuristic guided by three different non-compensatory, one-reason decision making strategies, on two classes of CSPs: < 30, 8, .26, .66> and < 30, 8, .12, .5>. Time (in seconds), errors and nodes are per problem. Figures in bold represent a statistically significant improvement over ACE without recognition.

Class	Criterion	ACE		Minimalist		Take the Last		Take the Best	
30-8-.26-.66	Overall time	3.10	(3.04)	**2.72**	(2.57)	**2.85**	(2.73)	**2.37**	(2.19)
	Tier 3 time	1.41	(1.51)	**0.70**	(0.62)	**0.75**	(0.68)	**0.61**	(0.68)
	Errors	105.11	(107.64)	119.12	(134.69)	116.97	(129.27)	113.77	(127.33)
	Nodes	165.11	(107.64)	179.12	(134.69)	176.97	(129.27)	173.77	(127.33)
30-8-.12-.5	Overall time	0.76	(0.57)	**0.71**	(0.49)	**0.71**	(0.42)	**0.66**	(0.44)
	Tier 3 time	0.44	(0.38)	**0.37**	(0.31)	**0.37**	(0.28)	**0.36**	(0.34)
	Errors	13.80	(15.52)	13.38	(15.26)	14.12	(16.17)	12.97	(14.36)
	Nodes	73.80	(15.52)	73.38	(15.26)	74.12	(16.17)	72.97	(14.36)

tional strategy (Take the Best) outperformed the other two.

The next class of problems on which we tested this approach was <30, 8, .12, .5>, a somewhat easier set. Here again, all three strategies achieved significant speedup, this time without increasing the number of errors or the size of the search tree. Finally, we tested our approach on a relatively easy class, <30, 8, .1, .5> (results not shown) where no changes could be detected.

Discussion

ACE is *complete*, that is, as constructed it is guaranteed to solve any solvable CSP — eventually. Expertise, however, requires that one solves problems efficiently. (All solutions to a CSP are defined to be equally good. When CSP researchers talk about *optimal* search, they refer one that does the least work, as measured by propagation.) Although extensive computation can minimize, or even eliminate, incorrect value selection, such computation may simply not be worth the time. Indeed, a solver that makes many inexpensive mistakes may actually arrive at a solution more quickly, despite a somewhat larger search tree. In this sense, ACE is a satisficer — it makes "good enough" decisions (Simon, 1981). Even when fast and frugal reasoning introduces additional error, the program solves problems faster. Both satisficing and our implementation of recognition, it should be noted, are tolerable only on problems where errors are relatively harmless.

The results of these experiments indicate that enhancing an intelligent and ecologically rational system with fast and frugal reasoning can save computation time, but is not guaranteed to do so. Because recognition, as we have implemented it, is a consequence of previous errors on the current problem, performance on these three problem classes requires individual explanations. On the relatively easy problems of <30, 8, .1, .5>, fast and frugal reasoning does not improve performance because there are not enough retractions during search to support subsequent recognition.

Fast and frugal reasoning that is also ecologically rational (i.e., Take the Best) provided more speed-up here than the other strategies. Recognition serves as a filter for the best of the options; it recycles earlier reasoning inherently. Nonetheless, ACE still needs to select from among the recognized options the one which is likely to be the most productive, and Take the Best is one way to do that.

On the problems of medium difficulty of <30, 8, .12, .5>, fast and frugal reasoning achieves speedup because it avoids some repeated computation. It also does so without significantly introducing more error, because recognition forces persistence by attempting to restrict ACE to previously-chosen options. Even if these "recycled decisions" are wrong, there are simply not enough wrong ones to introduce many new retractions.

On the very difficult problems of <30, 8, .26, .66>, Take the Best introduces significantly less error and does less work than the other fast and frugal strategies do. The additional errors on these problems suggest that accurate decision making here is more subtle and complex than a single heuristic can support, and certainly more than recognition alone can handle. Even without fast and frugal methods, ACE makes more mistakes solving these problems, simply because the problems are harder, Therefore there are more errors that may be recycled by recognition, as well as simply more recognized options to choose from. Take the Best introduces significantly fewer errors because it uses ecological rationality to avoid recycling some of them. Here again the speedup is achieved through the tradeoff between inexpensive errors and savings in computation time.

ACE's version of recognition is not the situation-based recognition described in (Klein & Calderwood, 1991). In that work, particular features of a situation bring to mind possible solution approaches, approaches that may require adaptation for the current situation. There, recognition may be paraphrased as "I was once in a similar situation where this sequence of decisions worked well, so I will see if I can adapt it to work again here, testing it first in simulation." ACE's recognition, in contrast, may be paraphrased as "I have seen that option before, considered it carefully (perhaps with different alternatives and in a somewhat different context at the time), and have decided to prefer it once again, without considering any potential consequences." ACE's recognition applies only to a single decision, not to a sequence; it does not permit adaptation; it makes no situation assessment; and it relates to previous experience in the same problem.

To make use of fast and frugal reasoning, as we have implemented it, a system needs to have a body of heuristics with which to make decisions and, if it is to take the best, it needs a metric on those heuristics. Furthermore, since recognition, as we have interpreted it, requires decisions that are made under some erroneous circumstances and then withdrawn (gray nodes in Figure 2), the system must not have perfected decision making, or there will be no prior, within-problem decisions to recycle. Thus, fast and frugal reasoning can improve mediocre or even fairly reputable performance, but cannot improve flawless performance, for without errors there can be no recognition.

Future Work and Conclusion

Previously-made decisions are here recycled through the recognition heuristic. Whether or not ACE would have remade those decisions, and in the same sequence, without recognition remains to be determined. Future work will examine ACE's decisions at the same level but in different branches of the search tree, with and without fast and frugal reasoning.

Although we did not use it in these experiments, ACE can partition its tier-3 heuristics into any number of subclasses. We intend to compare ACE's performance with different numbers of tier-3 subclasses to ACE's performance with Take the Best, which can be thought of as a "one heuristic to a subclass" partition.

Fast and frugal reasoning has been shown here to have a significant impact on an already competent CSP solver. The premise that attractive options remain attractive as problem solving progresses enables at least this program to solve problems better. Furthermore, the problems we have used here are sufficiently general to suggest that our results have a potentially broad impact. We are optimistic that, in problem areas that tolerate errors, fast and frugal reasoning, as implemented here, can make an important contribution to problem solving.

Acknowledgments

This work was supported in part by NSF IIS-0328743 and by PSC-CUNY. Thanks for their support in this work go to Gene Freuder, Anton Morozov, Rick Wallace, CUNY's ACE study group, and the Cork Constraint Computation Centre, supported by Enterprise Ireland and Science Foundation Ireland.

References

Bessière, C., & Régin, J.-C. (2001). Refining the basic constraint propagation algorithm. *JFPLC*, 1-13.

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.

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

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

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

Epstein, S. L., Freuder, E. C., Wallace, R., Morozov, A., & Samuels, B. (2002). The Adaptive Constraint Engine. In P. Van Hentenryck (Ed.), *Proceedings of CP2002* (Vol. LNCS 2470, pp. 525-540). Berlin: Springer Verlag.

Gent, I. E., MacIntyre, E., Prosser, P. and Walsh, T. (1996). The Constrainedness of Search. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 246-252).

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

Gigerenzer, G., & Selten, R. (2001). *Bounded Rationality: The Adaptive Toolbox*. MA: MIT Press.

Gigerenzer, G., Todd, P. M. & The ABC Research Group (1999). *Simple Heuristics that Make Us Smart*. NY: Oxford University Press.

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

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

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

Mackworth, A. K. (1977). Consistency in Networks of Relations. *Artificial Intelligence*, 8, 99-118.

Novick, L. R., & Coté, N. (1992). The Nature of Expertise in Anagram Solution. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, IN.

Oppenheimer, D. M. (2003). Not so Fast! (and not so Frugal!): Rethinking the Recognition Heuristic. *Cognition*, 90, B1-B9.

Sabin, D., & Freuder, E. C. (1994). Contradicting Conventional Wisdom in Constraint Satisfaction. *Proceedings of the Eleventh European Conference on Artificial Intelligence*, Amsterdam.

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

Simon, H. A. (1981). *The Sciences of the Artificial* (second ed.). Cambridge, MA: MIT Press.