In Support of Pragmatic Computation

Susan L. Epstein
Department of Computer Science
Hunter College and The Graduate School of The City University of New York
susan.epstein@hunter.cuny.edu

Abstract

This paper argues that human cognition is structured to support solving in unmanageably large problem spaces. Machines face similar challenges, and can therefore benefit from the same kinds of approaches. Pragmatic human cognitive mechanisms already appear in AI artifacts, and more should be considered. Examples are drawn from human and programmed approaches to game playing, constraint satisfaction, and protein sequence alignment.

People and artificially intelligent machines both solve problems. From a machine's perspective, however, the ability of people to function not only adequately but expertly is nothing short of miraculous. Human memories are clearly limited and often inaccurate, human algorithms are mysterious, and human heuristics are naïve. Although endowed with good sensory mechanisms, people often choose to ignore relevant data. Although they invented logic, they rarely represent their reasoning that way. The issue is whether an artificial intelligence should capitalize on what is known about human reasoning.

It is the thesis of this paper that cognitive science has been, and must continue to be, relevant to the way AI programs solve problems. This is not because people are an exemplar of cognition, but because human cognitive mechanisms were developed to cope with problems that are beyond our computational capacity. People do *pragmatic computation*, that is, they arrive at and explain adequate decisions efficiently. Here, an adequate decision is one that supports the multiple goals of the decision maker, and efficiency is measured by the consumption of cognitive resources. Efficient explanation entails a concise description of the process, possibly for the benefit of other agents. When people ignore information or modify a procedure in a way that compromises its correctness because they can do so with impunity, they are computing pragmatically. When an artificial intelligence is pressed beyond its capacity, it should compute pragmatically too.

Real-world decisions are always subject to a myriad of potential but unlikely catastrophes that lurk in a dynamic environment. Factoring each of them into a decision (e.g., crossing the street) would consume additional computing resources, but likely produce the same decision. Because, in the real world there are always additional goals to address, an intelligence, real or artificial, would do better to compute pragmatically and move on to the next problem.

This paper describes some pragmatic human cognitive mechanisms, and shows how many programs (even "brute force" ones) already incorporate them. After some basic definitions, it explores how people and programs use knowledge, representation, and meta-heuristics in problem solving. It considers the role of visual cognition and self-awareness, and poses some challenging issues for research.

Throughout, examples are drawn from game playing, constraint satisfaction, and protein sequence alignment. A *constraint satisfaction problem* <*X,D,C*> is a set of variables *X*, a set of domains *D* for those variables, and a set of constraints *C*, restrictions on the way values from the domains may be assigned to the variables. Constraint solving seeks an assignment of a value to each variable from its domain so that all the constraints are satisfied. Graph coloring and scheduling are both constraint satisfaction problems. *Protein sequence alignment* seeks to arrange complex physical objects in a way that emphasizes their three-dimensional and/or evolutionary similarities. Given a string on a 20-letter alphabet (the *residues*), there is usually a single way the protein it represents will arrange itself in space. A good alignment of two such strings, for example,

S K — T T W T
T K S S T W S

pairs identical, biophysically or evolutionarily similar residues with each other. Each gap insertion (here the symbol —) incurs a penalty, part of which is proportional to its length. Programs for constraint satisfaction and protein sequence alignment are powerful but still inadequate for many tasks; both problems are NP-complete.

Problem Solving and Search

First, some basic definitions. A *world state* is a set of data that describes some limited perspective at a moment in time, such as pieces on a chess board or assignments of values to constrained variables. An *operator* is a function that transforms one world state into another. A *goal test* is a boolean function on world states, and a *goal state* is one that passes the goal test. A *problem* consists of a set of world states (the problem's *state space*), a set of operators defined on them, and a goal test. *Problem solving* is the search for some world state that passes the goal test. The *global search* paradigm searches state space from a designated world state (the *initial state)*, applying one operator at a time until some world state passes the goal test. A *solution* to a problem is either a goal state (e.g., constraint satisfaction or protein sequence alignment) or the search path to it (e.g., chess).

In a state space of challenging size, intelligence requires more than fast global search — it also requires knowledge. For example, a machine cannot play chess by exhaustive lookahead because the chess state space is unmanageably

large. Although Deep Blue played chess on special-purpose hardware that examined millions of states per second, it still lost its first match against Gary Kasparov in 1996. As computers' speed and memory increase, the goals set for AI programs become more ambitious. Thus there will always be problems (e.g., Go, scheduling, protein sequence alignment) on which uninformed search is inadequate.

The Role of Knowledge

Expert knowledge focuses, directs, and resolves search. At the beginning of search, a problem solver's initial decisions (its *opening*) prune away large portions of the space. An opening is a knowledge-based heuristic; its function is to make the state space more manageable without dramatically reducing the likelihood that a solution can be found. In constraint solving, for example, openings are crucial; there is recent evidence that one need only get the first few variables' values correct to solve certain kinds of very large problems (Gomes, Selman et al. 2000). A game opening is simply a sequence of moves known to lead to successful play in the hands of an expert. Although carefully chronicled and studies, game openings are heuristics that may be revised or even discredited.

Even in large state spaces, it may be possible to search exhaustively for a path to a goal state beginning from what we call *solvable states*. Solvable states arise relatively deep in search, and may not form part of the eventual solution. As protein alignment decisions are made, for example, short sequences arise that must be aligned; as values are assigned in constraint solving, small subproblems arise. Both can be solved by exhaustive search. Unlike openings, knowledge about solvable states is guaranteed correct because it is based on goal tests for all the states in some portion of the space. If solvable states recur, either in the domain (as they do in chess) or repeatedly in a single problem (as they do in constraint solving), storing their results can speed search. Traditionally, solvable state libraries for games (*endgame* knowledge) have been developed by hand and shared among human experts.

Although a human expert can get a reasonable head start from an opening, and search flawlessly from solvable states, in between she must still make search decisions. A heuristic *evaluation function* estimates the likelihood that search progressing through a state will lead to a solution. An evaluation function is based on *features* (state descriptions) that predict that likelihood. Chess players think about pawn structure and control of the center, biophysicists about hydrophobicity and beta-sheets, and constraint solvers about promise and fail-firstness. A human expert may use tens (or even hundreds) of such features.

Many AI artifacts use knowledge the way people do. One can engineer a problem solver for a particular domain by providing a search engine with openings, a solvable state library, and a good evaluation function. Openings are provided by programmers. Computers have verified knowledge about solvable states and, with their accuracy and speed, extended it (Hsu 2002; Schaeffer, Bjornsson et al. 2003). A programmer usually provides an AI artifact with the features for its evaluation function, although the program is generally called upon to find a method to combine them.

When Deep Blue won its rematch against Kasparov in 1997, the most significant improvement in the program had been an infusion of grandmaster-level knowledge. At the time, there was much discussion about the fact that the program was not playing the way people do: it did not plan or learn, it just did fast, deep search. What the press overlooked, however, was that Deep Blue had inherited from its human creators the same knowledge-based coping mechanisms people use when confronted with a problem of unmanageable size. The winning version of Deep Blue included an enlarged database of openings curated by three grandmasters plus an extensive, computer-generated endgame database. Most importantly, the winning version had an elaborate evaluation function carefully tuned under the supervision of a fourth grandmaster.

Other game-playing champions continue in this tradition. Chinook's creators persist in their drive to solve checkers, through a combination of intensive state-space exploration and the management of an enormous endgame database (Schaeffer, Bjornsson et al. 2005). TD-gammon the backgammon player (Tesauro 1994) emphasized learning to combine the features in its evaluation function. Logistello, the Othello program that defeated the human world champion in 1997, uses on a combination of pattern-based, learned features for its evaluation function (Buro 1999). All of these programs rely on extensive opening databases and have had considerable human guidance on the features for their respective evaluation functions. (Endgame databases are constructed from all possible positions. Because the number of pieces on its board increases as play progresses, Logistello has no endgame database.)

Despite a plethora of domain-specific heuristics, very strong constraint solvers and protein sequence aligners have yet to emerge. There is little opening knowledge for those domains, only the certainty that openings are disproportionately important. When knowledge is unavailable, other methods, discussed below, may be brought to bear.

In summary, many state-of-the-art game-playing programs are built the way people play because it is a pragmatic way to address an unmanageably-large state space. These programs carry things further than people do: more openings, larger and more accurate endgame databases, faster and deeper search, more feature combinations tested to select a good balance. Nonetheless, their evaluation function features and their openings come from people, and are therefore based on human experience. Of greater concern is the programs' inability to reason about the state space itself.

Search Meta-heuristics

Search meta-heuristics offer search control at a higher level than an evaluation function. A search meta-heuristic

represents information about the state space to facilitate decisions. For example, rotational symmetry plus the rules make all corners of the tic-tac-toe board equivalent. Similarly, any color may be chosen for the first vertex during graph coloring.

One search meta-heuristic to manage a large, repetitive state space is to augment search with guidelines. "Open in the center" or "align the W residues" prunes search dramatically. A *strategy* is a set of guidelines that summarizes how to make decisions. For example, *lose tic-tac-toe* is a game identical to tic-tac-toe except that the goal test is negated, so that the objective is avoid three in a row, column, or diagonal. Expert lose tic-tac-toe play requires different strategies for each competitor.

On a larger scale, there are thousands of proverbs for the game of Go, such as "beware of going back to patch up" or "don't go fishing while your house is on fire." These proverbs are descriptions of how centuries of expert-level play has proceeded. Like openings, proverbs are heuristic; they serve as a synopsis of the way search has worked for many experts. A correct strategy or proverb is a synopsis of the way the state space is constructed.

One way to describe parts of a state space is with a *role*, a model one agent has of the beliefs, behavior, and experience of another. People model each other as part of their social interactions because it permits them to anticipate the portions of the state space in which they are likely to find themselves. This is often a response to the large branching factor inherent with imperfect information (e.g., Scrabble or poker), where competitors must weigh knowledge and assess risk. Game-playing programs have capitalized on this cognitive approach too. MAVEN, a strong Scrabble playing program, anticipates the moves of its opponent (Sheppard 2002). Roles are particularly important in games like poker, where deliberate misinformation is often part of the world state. (Billings, Burch et al. 2003).

It may not be clear how to implement a search meta-heuristic, even when one can argue for its importance. Consider, for example, a problem solver to color a large graph. As each node is colored, all edges to its neighbors are removed and the possible colors of its neighbors are reduced. A typical search heuristic, (e.g., "color the node of maximum degree") hops about the graph, coloring nodes and removing edges. After enough nodes have been colored, an additional feature becomes important: connectedness. Clearly, search should be restricted to one connected component at a time until that component is known to be solvable. Less clear, however, is which newly-disconnected component should be searched first.

Rather than search rigidly, people can reason about the structure and content of a state space to curtail unnecessary search. For example, it is possible to postpone coloring any tree that arises until all the cyclic components have been colored. (There is a test linear in the number of tree nodes for solvability.) That way, if an earlier color assignment was wrong, the work that would have been done in the tree need not be undone. People often use this "as good as solved" approach to temporarily abandon a subgoal.

Search and Representation

Both AI researchers and cognitive scientists have confirmed the significance of representation during problem solving. When re-represented, problems become far more manageable and even prove to be special cases of more challenging problems (Amarel 1968; Anzai and Simon 1979). Take, for example, the mutilated checkerboard problem, where a checkerboard has had its upper left and lower right corner squares removed, so that only 62 squares remain. The task is to cover the checkerboard with 31 dominoes, each of which can be placed vertically or horizontally to cover two adjacent squares. Ostensibly this problem has a very large state space. The clever solver, however, will note that both the squares removed from the checkerboard were the same color, that a domino by its nature covers one red and one black square, and that therefore there can be no solution.

A good problem representation may reduce the size of the state space. The mutilated checkerboard problem is interesting because the "obvious" visual representation of the world states (as if one were placing dominoes) is the wrong one — what is needed is an abstraction that represents only the number of uncovered red and black squares. In this much smaller space, search is quick and clear.

Indeed, under a good representation, reasoning about the state space can completely eliminate the need for search. For the mutilated checkerboard, a person is likely to declare the problem unsolvable without any search at all: the initial state is (32, 30), the only operator reduces the state (x, y) to $(x\text{-}1, y\text{-}1)$, $|x - y|$ remains invariant, and (0,0) is unreachable.

Multiple representations may be dictated by solution criteria or by search heuristics. Multiple solution criteria may require different representations of the problem state. For example, protein sequence alignment is judged based on the pairing of individual residues, as well as on the alignment of the proteins' secondary and tertiary structures. Each representation should therefore be considered as the alignment is constructed. A solver for such a problem must work with all relevant representations during search. Nonetheless, the integration of multiple representations requires thoughtful implementation.

Significantly, multiplicity is a hallmark of human cognition. People are equipped with a variety of sensors that provide many kinds of information, carried in a variety of formats. Human memories are redundant, with multiple paths to access the same information. People entertain and use multiple representations and decision-making heuristics simultaneously (Biswas, Goldman et al. 1995). In a program, it is possible to replicate this diversity and manage it effectively and efficiently. The FORR architecture, for example, has supported domain-specific applications for game playing (Epstein 2001), simulated path finding (Epstein 1998), and constraint solving (Epstein, Freuder et al. 2005). These programs learn to balance many domain-specific, decision-making heuristics, each of which references some subset of the available representations. No uniformity is imposed on the heuristics; representations are

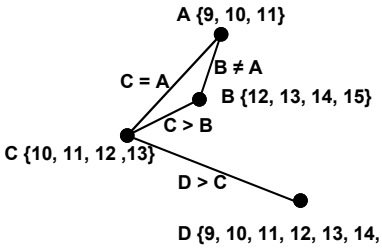

Figure 1: The underlying constraint graph for the tennis problem in the text. Vertex labels are variables (initials of first names) with 24-hour domain values as starting times; edge labels are binary constraints.

shared and computed only on demand. Each program learns to balance its heuristics and learns to achieve expert performance in a wide variety of problem classes.

Visual Cognition

Many real-world tasks have a strong visual component. When great mathematicians look back on their discoveries, they often describe the moment of recognition in visual terms, complete with colors or flashing lights (Hadamard 1945; Pascal 1964; Poincaré 1970). Reading such memoirs, one becomes convinced that logic is the language of argumentation, rather than the language of discovery.

If a task does not have a strong visual component, people often add one to facilitate their decisions. Consider, for example, the following constraint problem. "A tennis court is available from 9 AM and 4 PM. Alice, Bob, Cindy and Doug all want to play. Alice can only play in the morning, Bob can only play in the afternoon, and Cindy can only play between 10 AM and 2 PM. Doug is always available. Everyone wants to play singles for 1 hour. Bob will play only with Cindy or Doug. Doug wants to play after Cindy. Cindy wants to play with Alice and after Bob." Figure 1 represents the relationships in this problem with a diagram called a *constraint graph*. Most variable-selection and value-selection heuristics in the constraints literature use properties of some diagram like this, even though the problem itself is abstract and has no visual features at all. Such a representation removes extraneous detail (e.g., the sport, the names, the times of day). It also summarizes the remaining information concisely, and makes interrelationships (and their absence) clear.

Diagrams are often powerful adjuncts to human reasoning, but AI programs still require human guidance to exploit them. A geometry theorem prover developed search openings by "looking" at an accompanying diagram for each problem to find likely openings (Gelernter 1963). A representation language for the relationships among objects in space supported powerful reasoning about complex molecules in three-dimensional space (Glasgow 1993).

Indeed, a good diagram of the state space itself offers considerable explanatory power. For example, Figure 2 diagrams the state space for a simple game (Epstein 2005). A game is said to be *strongly solved* if, from every possible state, an efficient algorithm is known that determines the value of the position. Effectively, once a game is strongly solved, every state is a solvable state. Although the game in Figure 2(a) is simple, the state space diagram in Figure 2(b) produced by a cognitive science student generated a set of guidelines that strongly solved the game. The method has been extended to other games as well.

Finally, cognitive scientists study the impact of cues on human problem solving. The problems may be far simpler but the issue is the same: how to focus attention on the salient features of the problem in a complex environment. Move generation in game playing is a good example of this. Human chess experts report considering about six or seven possibilities from any position, even though there are likely to be three times that many legal moves. Go masters, according to measurements of their eye movements, consider only three or four moves from among hundreds (Yoshikawa and Saito 1997). Among the Go proverbs are many visual cues, such as "when your opponent is thick, you must also become thick" or "grab the fourth point of the bamboo joint." Like the other proverbs, they summarize some aspect of the state space, but they offer visual cues that focus attention as well. Visual cognition interleaves with high-level reasoning in complex ways.

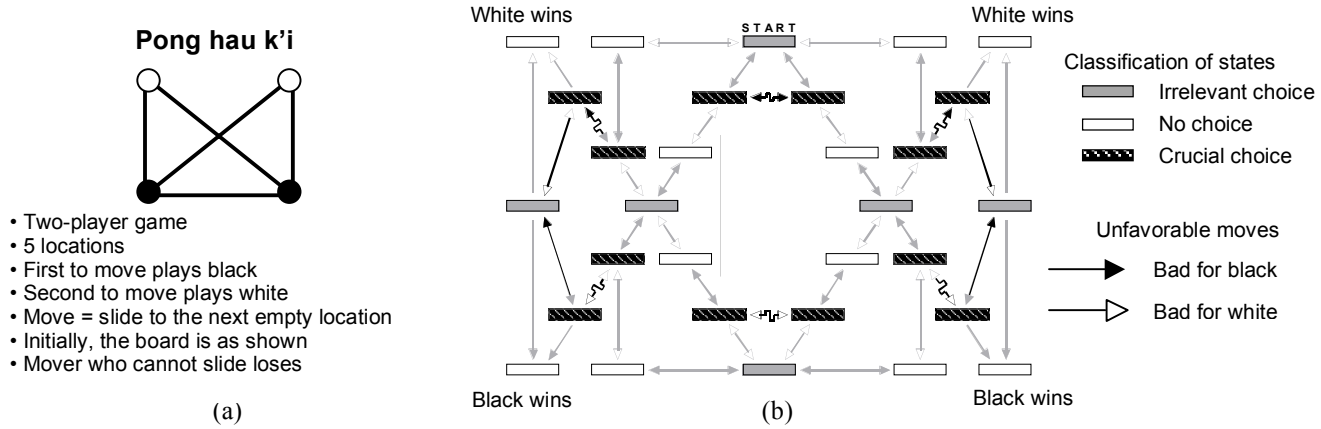

Figure 2: (a) A simple game and (b) a strategic explanation of its state space, where dark nodes are crucial decision-making states. Black arrowheads are bad moves for black; white arrowheads are bad moves for white.

Ignoring Knowledge

Surprisingly often, people rely on a single heuristic to make good enough decisions. Many psychologists study such *fast and frugal* reasoning (Gigerenzer, Todd et al. 1999). Fast and frugal reasoning methods are believed to rely on *recognition*, the favoring of familiar objects over unfamiliar ones. A fast and frugal reasoner selects only among recognized choices, and then applies a second standard, such as "try the last heuristic used." Although at first glance fast and frugal reasoning seems draconian, it works surprisingly well on challenging constraint problems (Epstein and Ligorio 2004).

In many domains, it is possible to solve large problems quickly through reasoning that deliberately overlooks relevant features of the problem. Consider two algorithms for solving a large constraint satisfaction problem. Algorithm 1 uses a single heuristic that focuses on variables with a low value for a particular property and assigns values to them at random. Algorithm 2 uses the heuristic of Algorithm 1 but also has a reasonably accurate value selection heuristic. One would expect that the number of mistakes made by Algorithm 1 would be intolerable, but it is actually good enough to solve many problems quickly, even though it is often wrong. Algorithm 2, although it makes far fewer errors, is unacceptably slower on many problems. If an *expert* is indeed one who solves problems faster and better than the rest of us (D'Andrade 1990), then Algorithm 1 is (disturbingly) more expert. Its decision making is good enough to find a solution as long as errors are relatively inexpensive. Pragmatic computing, recall, makes good enough decisions efficiently. By this standard, Algorithm 1 would be preferred to Algorithm 2. Similarly, highly-accurate game-playing features for an evaluation function are often costly to compute, so that a few of them (Berliner and Ebeling 1989) may be the equivalent of many less incisive ones (Berliner 1980).

Self-awareness

People attend to and evaluate their own problem solving process. Their *self-awareness* critiques search and acts to improve problem solving. For example, self-awareness helps people decide to start over on a problem. There is a substantial AI literature on restart. For example, on very large problems, researchers have found a surprisingly simple and effective way to randomize global search on constraint satisfaction problems. It uses a simple heuristic to select among variables, breaking ties at random, and it selects values at random, but when some difficulty arises (e.g., four value assignments retracted during search), it begins search again (Gomes, Selman et al. 2000). Although it may require many restarts, this process has proved remarkably efficient on a variety of difficult problems.

While solving a set of problems, a person may also recognize that the learning process itself is not going well and begin again. Although it is more difficult for people than for machines, it is possible to wipe clean the knowledge derived from experience and take a fresh look at similar problems. *ACE* is a constraint solving program that learns to combine multiple heuristics to solve constraint satisfaction problems. When problem difficulty is non-uniform, particularly easy problems can mislead ACE, so that occasional runs do not acquire a good mixture. We have successfully adapted the program to monitor its own prowess on a sequence of problems and begins a new sequence if its skill is not satisfactory (Petrovic and Epstein 2006).

At some point in many difficult problems, it is clear that the hard decisions have been made, that is, that subsequent incorrect decisions will have a trivial impact. At this point, people generally shift to a less-computationally intensive method. ACE has been able to learn that point for a class of constraint satisfaction problems, and curtail computation after it (Epstein, Freuder et al. 2005). The resultant speedup is another example of pragmatic computing.

Research Challenges

Despite the successful incorporation of some human cognitive processes into AI programs, many research challenges remain. In particular, cognitive science still has much to learn about focus of attention, addressing multiple goals, and interleaving multiple representations.

Vision can be a hindrance as well as a help. In one cognitive science experiment, for example, people were asked to play several games, all of which were well-disguised isomorphs of tic-tac-toe (Zhang 1997). Although each game had the same number of operators and the same number of world states, some were far more difficult for people than others. People applied some operators and manipulated some state representations far more slowly than others, just the way machines do. All of the isomorphs represented the game pieces with more complex symbols than X and O (e.g., colors or shapes) and represented their alignment in a more complex, non-linear way (e.g., territory). Clearly the extraneous visual detail was a handicap.

Although learning is an essential component of intelligence, the biases it produces can be a hindrance. For example, people find lose tic-tac-toe far more difficult than tic-tac-toe. This is because they attempt to transfer their experience with the ordinary version by doing exactly the opposite in lose tic-tac-toe. A program with an ordinary search engine seems a better alternative because, without that initial bias, it would play both games the same way. Nonetheless, people eventually develop strategies for these games, arguably a more pragmatic computation.

Multiple representations have their perils too. Little is understood yet about how people successfully integrate them, or how people develop efficient representations. Multiple representations are often detected only after brain injuries, where they provide tantalizing glimpses of alternative storage and retrieval mechanisms.

People satisfice either because they cannot compute fast enough to include everything they know in their computations or because there are other problems to be solved. Although they extend human approaches beyond people's

ability to calculate and remember, many of the programs of which AI is most proud are still modeled on how people address large problem spaces. These programs may not emulate human thought at low levels, but they borrow heavily, not only from human expert knowledge, but also from human pragmatic knowledge about problem solving. Their "good enough" solutions must be acceptable because, for some problems, computers will never be large enough and fast enough to achieve any other kind. The issues for cognitive science and AI researchers are clear:

• Given low-level sensory data, how do people (or solvers) select information to represent world states?

• Given a host of choices, how do people (or solvers) focus their attention?

• Given relevant features that describe choices, how do people (or solvers) combine or choose among them?

• Given a problem solving situation, how do people (or solvers) develop new heuristics?

Acknowledgements

This work was supported in part by NSF IIS-0328743. Thanks to CUNY's FORR study group and the Cork Constraint Computation Centre, led by Gene Freuder, for their continued support of this work.

References

Amarel, S. (1968). On Representations of Problems of Reasoning about Actions. *Machine Intelligence 3*. D. Michie. Edinburgh, Edinburgh University Press**:** 131-171.

Anzai, Y. and H. Simon (1979). The Theory of Learning by Doing. *Psychological Review* **36**(2): 124-140.

Berliner, H. and C. Ebeling (1989). Pattern knowledge and search: The SUPREM architecture. *AIJ* **38**(2): 161-198.

Berliner, H. J. (1980). Backgammon Computer Program Beats World Champion. **14**(2): 205-220.

Billings, D., N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg and D. Szafron (2003). Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. *IJCAI-2003*, 899-905.

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

Buro, M. (1999). How Machines have Learned to Play Othello. *IEEE Intelligent Systems Journal* **14**(6): 12-14.

D'Andrade, R. G. (1990). Some Propositions about the Relations between Culture and Human Cognition. *Cultural Psychology: Essays on Comparative Human Development*. J. W. Stigler, R. A. Shweder and G. Herdt. Cambridge, Cambridge University Press**:** 65-129.

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

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

Epstein, S. L. (2005). Thinking through Diagrams: Discovery in Game Playing. *Spatial Cognition IV*, Springer-Verlag, 260-283.

Epstein, S. L., E. C. Freuder and R. M. 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. *Cognitive Science 2004*, Chicago, Lawrence Earlbaum, To appear.

Gelernter, H. (1963). Realization of a Geometry-Theorem Proving Machine. *Computers and Thought*. E. A. Feigenbaum and J. Feldman. New York, McGraw-Hill**:** 134-152.

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

Glasgow, J. I. (1993). The Imagery Debate Revisited: A Computational Perspective. *Computational Intelligence* **9**(4): 309-333.

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

Hadamard, J. (1945). The Psychology of Invention in the Mathematical Field. New York, Dover Publications.

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

Pascal, B. (1964). *Pensées de Pascal*. Paris, Éditions Garnier Frères.

Petrovic, S. and S. L. Epstein (2006). Full Restart Speeds Learning. *FLAIRS-2006*, In press.

Poincaré, H. (1970). *La Valeur de la Sciènce*. France, Flammarion.

Schaeffer, J., Y. Bjornsson, N. Burch, A. Kishimoto, M. Muller, R. Lake, P. Lu and S. Sutphen (2005). Solving Checkers. *IJCAI-05*,

Schaeffer, J., Y. Bjornsson, N. Burch, R. Lake, P. Lu and S. Sutphen (2003). Building the Checkers 10-Piece Endgame Databases. *Advances in Computer Games X*, Kluwer Academic Publishers**:** 193-210.

Sheppard, G. (2002). World-championship-caliber Scrabble *Artificial Intelligence* **134**(1-2): 241-275.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master level play. *Neural Computation* **6**(2): 215-21.

Yoshikawa, A. and Y. Saito (1997). Hybrid Pattern Knowledge: Go Players' Knowledge Representation for Solving Tsume-Go Problems. *South Korean International Conference on Cognitive Science*,

Zhang, J. (1997). The nature of external representations in problem solving. *Cognitive Science* **21**(2): 179-217.