Integrating a Portfolio of Representations to Solve Hard Problems

Susan L. Epstein

Hunter College and The Graduate Center of The City University of New York
susan.epstein@hunter.cuny.edu

Abstract

This paper advocates the use of a portfolio of representations for problem solving in complex domains. It describes an approach that decouples efficient storage mechanisms called *descriptives* from the decision-making procedures that employ them. An architecture that takes this approach can learn which representations are appropriate for a given problem class. Examples of search with a portfolio of representations are drawn from a broad set of domains.

Representation and cognition

A *representation* formally describes objects and their relationships, and provides ways to use them. In a complex domain, however, there may be many plausible representations for the same information about the search space and an agent's position in it. Reasoning in such a domain may not be well served by a single representation. This paper proposes a *portfolio* of representations instead, one that takes different perspectives on the same objects and highlights different relationships among them. Its principle result is the automated coordination of such a portfolio to make good decisions during problem solving.

At the atomic level, every representation has the same facts about the task at hand. In a board game, for example, the atomic level itemizes the location of each playing piece and whose turn it is to move. The power of a representation comes from its ability to view those facts in a way that supports expert reasoning. In chess, for example, pawn structure describes an important offensive and defensive mechanism. The facts, where the pawns are on the board, are no different. The difference lies in the aggregation of those facts and how that aggregation can be used.

To solve problems better, people often shift from one representation to another. A novice eventually solved the Tower of Hanoi expertly with her third representation (Anzai and Simon, 1979). A mathematician solved a generalized version of the Missionaries and Cannibals problem with a sequence of increasingly powerful representations (Amarel, 1968). More recently, graduate students in cognitive science documented their own representational shifts while learning to play a simple game (Epstein, 2005). In each case, only one representation was judged ideal.

In some domains, however, experts maintain a portfolio of representations, and select an appropriate one. For example, geneticists represent a protein as a string of nucleotides, a string of amino acids, or a three-dimensional structure. A representation may sacrifice detail for clarity. Amino acids, for example, are shorthand for nucleotide triples (*codons*), but some amino acids have more than one codon. As a result, comparison of nucleotide strings may detect a mutation that would go unnoticed in amino acid strings. In turn, amino acid strings simplify the detection of repetitive patterns. Thus, different representations better serve different kinds of analysis.

Representations need not be employed one at a time. An expert chess player, for example, considers more than *material* (the number and value of playing pieces) to select the next move. Control of the center of the board, possible *forks* (multiple threats), and defensive strategies are likely to temper the drive to conserve and amass material.

The premise that people employ multiple representations simultaneously is supported by their reported use of multiple strategies to make a decision. For example, undergraduates playing simple board games reported the use of as many as seven strategies (Ratterman and Epstein, 1995). Among them, these strategies referenced individual locations, sets of locations aligned along a straight line, forks, and learned visual configurations of pieces. Moreover, the skilled players among them reported the use of more strategies than did the unskilled players. Thus, multiple representations appear to support more expert reasoning.

At any given decision point, multiple representations may not support the same action. For example, architects design plans for individual subsystems such as room layout, plumbing diagrams, and electrical plans (Schraagen, 1993). Within the footprint of the structure, each specifies information about the same building, but ignores the others. When overlaid on tracing paper, these representations often interfere with one another — wiring may intersect pipelines. Nonetheless, human architects choose to create and then resolve these conflicts. The individual representations' contributions to the quality of the solution apparently outweigh the cost of coordinating them.

Multiple representations come at a price. They require more computation, some of which may be repetitive. Moreover, since the most appropriate representations are not always known in advance, a system may need to learn which prespecified representations to use. The next two sections of this paper describe how multiple representations can be organized and applied effectively to make decisions. Then an architecture that automatically sifts through and combines a portfolio of representations is detailed, along with a discussion of the challenges that arise with this approach.

A portfolio of descriptives

A *descriptive* is a domain-dependent storage mechanism intended to conserve computational resources while it supports decision making. In game playing, for example, descriptives could include openings, material, and patterns formed by playing pieces on the board. A descriptive is computed only when its trigger indicates that its current value may be outdated. (For example, material does not change unless there has been a piece capture.) A descriptive is only computed on demand, so costly computations are performed only when necessary. Furthermore, a descriptive's value is shared, not recalculated, by the decision-making procedures that reference it.

One example of a problem solver that learns with a portfolio of spatial descriptives is *Ariadne*, a system for simulated robot pathfinding (Epstein, 1998). Ariadne's task is to move a robot through a two-dimensional maze without a map, going around obstructions from an initial location to a goal location. Ariadne detects *visible locations* around the robot in four orthogonal directions to the nearest obstruction (dotted Figure 1). Each decision Ariadne makes moves the robot in a straight line to one of those visible locations. Instead of a map, Ariadne has a portfolio of descriptives that capture the robot's experience, shown in Figure 1.

Many of Ariadne's descriptives aggregate locations. *Corridors* are width-one passageways. They include *dead ends* (a single exit), *hallways* (two exits), and *pipes* (hallways whose endpoints lie in the same row or the same column). *Chambers* and *bottles* are room-like areas; the former is learned deliberately, with exploration, the latter afterwards. Each is a rectangular approximation of a restricted space with a single known access point. Finally, a *barrier* is a linear approximation of a wall that obstructs movement.

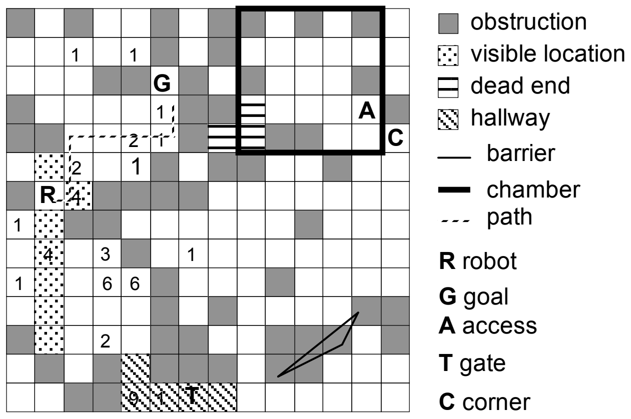

Figure 1: Examples of descriptives learned for one of Ariadne's mazes. A single decision moves the robot to any visible location. Numbers denote the frequency with which each location was learned as a base (shown for the left side only). The dotted line is a sequence of moves constructed from bases to move the robot near the goal.

Several other descriptives focus on properties of individual locations. A *corner* is the endpoint of a pipe that affords a 90° turn. A *gate* is a location that provides access between two quadrants. A *base* is a location that, although not in the direction from the robot to the goal, proved useful in a path that succeeded. Each base has a frequency that records how often it has served this way.

The values of all these descriptives are learned from experience in the same maze, much the way a person learns her way during travel around a new town. Bases and bottles are learned from traces of a successful task once it is completed. The others are discovered by Ariadne as the robot moves through the maze. When a descriptive's prespecified condition triggers, its value is recomputed. Some descriptives are recomputed after each decision, others before search or when search encounters difficulties.

A descriptive need not include all the atomic facts about the current state of the world. It provides a perspective, and that perspective may well be heuristic. For example, the rectangular approximation of the chamber in Figure 1 actually misses a few locations because the robot never ventured far enough to encounter them. Descriptives may also use other descriptives in their own computations.

Recall, however, that a representation is more than a storage mechanism; it must also provide ways to make use of the knowledge it encapsulates. The next section considers how to apply a descriptive to create a representation.

Applying descriptives as advice

The purpose of a representation is to support intelligent behavior on a *problem class*, a set of similarly characterized problems. Appropriate representations for a problem class support appropriate behavior there. In problem solving, appropriate behavior is a sequence of decisions that leads to a solution effectively. Thus a useful representation for problem solving is a descriptive used by at least one decision maker that consistently supports appropriate decisions and opposes inappropriate ones.

Assume that, like the people described earlier, a problem solver has a set of strategies with which to make decisions. (We defer the issue of their control to the next section.) Let us call these strategies *Advisors*, since they give advice at a decision point. A descriptive provides data in a storage structure that an Advisor can apply.

One Advisor can use the same descriptive in different ways. For example, if the goal is not within a particular chamber's rectangle, Ariadne's heuristic Advisor *Chamberlain* discourages individual decisions that would move the robot into that chamber. If the robot travels there despite this advice, *Chamberlain* also supports movement out of the chamber through its access point.

Different Advisors can also use the same descriptive. For example, Ariadne's Advisor *Outta Here* produces a short (at most 3-action) sequence to leave a chamber through its access point. An Advisor may also employ any number of descriptives. *Outta Here,* for example, uses both chambers and hallways to construct its action sequences.

Advisors should exploit descriptives to achieve problem-solving goals and to avoid pitfalls to solution. In some domains, the way to apply a descriptive is obvious. With Ariadne, for example, gates, bases, and corners facilitate movement through the maze, that is, they make more effective decisions when properly visited. Chambers, bottles, and barriers, on the other hand, obstruct movement and for the most part should be avoided. In other domains, however, the correct way to use a descriptive may vary with the problem class.

Consider, for example, constraint satisfaction problems (*CSPs*). Many difficult problems can be represented and solved as CSPs, including graph coloring, Boolean satisfiability, and planning and scheduling tasks. A CSP is defined by a set of variables, each with its own domain of possible values, and a set of constraints that restricts how subsets of those variables can hold values simultaneously. A *solution* to a CSP assigns a value from its domain to each variable so that it abides by all the constraints.

Search for a solution to a CSP is NP-complete. *Global search* assigns a value to one variable at a time, and tests to make certain that no constraint has been violated before it proceeds to the next assignment. If an assignment violates a constraint, backtracking returns to an earlier decision point to make a different assignment. (A variety of mechanisms exist to backtrack and to infer which untried values will fail; these are not our concern here.)

If a CSP is *binary* (has constraints only on pairs of variables), one popular way to visualize it is as a *constraint graph*, where each variable appears as a vertex and each constraint as an edge between the pair of variables it constrains. For a binary CSP, *density* is the fraction of possible constraints it includes and *tightness* is the average percentage of the possible value pairs its constraints exclude from the Cartesian product of the domains of their variables. When its density is very high or its constraints are very tight, a CSP is less likely to have a solution.

CSPs with low density and loose constraints generally have many solutions and are easy to solve. Very dense CSPs with tight constraints are also easy to "solve;" search quickly identifies some variable none of whose domain values can lead to a solution. For any given *problem size* (number of variables and domain sizes), the most difficult problems (those at the *phase transition*) have a density and tightness that permits few, difficult to find, solutions. To avoid a combinatoric explosion, difficult CSPs are addressed under a resource limit; a solver abandons a problem once it reaches that limit.

Unlike pathfinding, where the right ways to use a descriptive are clear, in constraint solving the right ways to apply a descriptive depends upon the problem class. There is an extensive literature of heuristics to select the next variable and its value during CSP search. The typical heuristic relies on a metric (readily implemented as a descriptive), and makes a decision by maximizing or minimizing the value of that metric over the actions. For example, the *degree* of a variable is the number of constraints in which it is involved. *Max Degree* is a heuristic that prefers variables with the highest degree. For many CSPs, *Max Degree* is an effective Advisor. There are, however, problem classes of CSPs on which *Max Degree* fails and its opposite, *Min Degree,* is effective.

In summary, the approach taken here views a representation as a descriptive plus the Advisors that reference it. The next section describes an architecture that supports such multiplicity, and learns which representations best guide decision making in a particular problem class.

Coordinating multiple representations

FORR (FOr the Right Reasons) is a cognitively plausible architecture for learning and problems solving (Epstein, 1992). A FORR-based system solves a problem with a sequence of decisions. Each decision selects an alternative from among the current available actions. Decisions are based upon advice from Advisors that reference descriptives. In this way, a FORR-based system integrates multiple representations.

Each FORR-based system has a *domain*, a set of problems (e.g., mazes or CSP classes). The FORR-based system for simulated pathfinding is Ariadne; the FORR-based system for constraint solving is *ACE* (the Adaptive Constraint Engine). To apply FORR to a domain, the user defines any number of descriptives and Advisors. Ariadne has 7 descriptives referenced by 31 Advisors; ACE has about 50 descriptives, referenced by more than 100 Advisors. Such multiplicity requires thoughtful organization.

An organizational hierarchy

FORR provides an organizational hierarchy of three tiers to coordinate all the Advisors in a domain. The user must place each Advisor in an appropriate tier based on its reliability, speed, and the nature of its advice.

Advisors in *tier 1* are always correct and provide their advice quickly. Advice from a tier-1 Advisor mandates a particular action or eliminates one or more actions from further consideration. For example, Ariadne's tier 1 Advisor *Victory* takes the robot directly to the goal if it is visible. Another tier-1 Advisor, *No Way,* prevents the robot from entering a dead end that does not contain the goal.

Within some time limit, an Advisor in *tier 2* formulates a sequence of actions. Each tier-2 Advisor has a situation-based execution trigger. For example, *Roundabout* is Ariadne's tier-2 Advisor to circumnavigate obstructions. *Roundabout* triggers when the robot is aligned with the goal but there is an intervening obstruction. *Roundabout* uses a pair of orthogonal directions to direct the robot closer to the goal, searching not along the wall but in ever-widening lateral swaths until it succeeds or its time has been exhausted. There is no guarantee that a tier-2 Advisor's action sequence will solve the subgoal it addresses.

An Advisor in *tier 3* is a heuristic whose advice is a set of comments. Each *comment* addresses an available action and assigns it a numerical *strength* that expresses the Advisor's degree of support for (positive strength) or opposition to (negative strength) that particular action. For example, *Giant Step* is Ariadne's tier-3 Advisor that prefers long moves. The further a visible location is from the robot's current position, the higher *Giant Step*'s comment strength will be for that location.

To make a decision, a FORR-based system presents a set of actions to its Advisors. For example, an action in Ariadne is to move to a visible location, and an action in ACE is to select a variable or a value for assignment. The decision is made in tier 1 if any Advisor there mandates an action or if among them they reject all but one. Otherwise, the remaining actions are forwarded to tier 2. If a sequence of actions is produced by any tier-2 Advisor, it is executed. Otherwise (typically about 95% of the time in with FORR), the tier-3 Advisors reach a decision a consensus that supports some remaining action. Advisors in tiers 1 and 2 are consulted in a prespecified order. In tier 3, however, every Advisor can comment on any number of actions at once.

Comments from tier-3 Advisors are combined in a process called *voting*. The simplest voting method sums the strengths of the comments that address each action, and then selects the action with the highest sum. Such even-handedness assumes that every Advisor is equally appropriate. For CSPs, however, that is clearly not the case. Instead, a FORR-based system can learn to manage tier 3.

Learning which representations are appropriate

An Advisor's tier suggests how appropriate its advice is likely to be. Advisors in tier 1 are expected to be quick and correct; their advice goes unquestioned. Advisors in tier 2 are allocated limited time and comment relatively infrequently. In contrast, tier 3 usually contains the vast majority of the Advisors, whose comments are frequent, possibly incorrect, and often in conflict with one another.

When a FORR-based system learns which tier-3 Advisors are the most appropriate for a problem class, it is also learning which representations (descriptives plus tier-3 Advisors) are the most appropriate. Every tier-3 Advisor in ACE, for example, has a dual Advisor; one maximizes a descriptive metric while the other minimizes it. On some CSP classes, one of a dual pair is more appropriate than the other; on other classes neither is appropriate. (We have yet to encounter a class where both are.)

The appropriateness of an Advisor is gauged by the impact of its comments on a completed task. Consider, for example, *Hoyle,* the FORR-based system that learns to play two-person, perfect information, finite-board games (Epstein, 2001). When Hoyle plays a *draw game* (one where the best possible outcome with error-free play on both sides is a draw), it should never lose. Advisors that supported decisions that led to a loss should therefore be penalized, while those that supported decisions that led to a win or draw should be rewarded. Hoyle learns after any contest at a game. Ariadne and ACE, however, learn only from solved problems, so failure to solve prevents learning.

Training instances to learn appropriate Advisors are taken from the trace of a task. They are individual decisions made by tier 3 during problem-solving search. Preprocessing the trace may clarify which decisions were sufficient for solution. For Ariadne, appropriate decisions move the robot to the goal; preprocessing removes loops from the traveled path. For ACE, *digressions* (subtrees eventually abandoned by backtracking) were wasted effort, so preprocessing removes most of each digression, retaining only the immediate actions that (mistakenly) led to it.

"Appropriate" is an approximation. It would be computationally exorbitant to determine precisely which move was responsible for the outcome of a non-trivial game. Similarly although the optimal path to Ariadne's goal or the optimal ordering for decisions during search for a solution to a CSP could be identified, the algorithms referenced here do not calculate them. Instead they assume that successful advice is worthy of attention, even emphasis. (Ramifications of this are considered in the next section.)

To assemble a portfolio of representations appropriate to a particular problem class, a FORR-based system learns weights for voting. Let $s(A_i, a, C)$ be the comment strength of tier-3 Advisor A_i on action a from among a set of actions C, and let w_i be the weight of A_i. Then the selected action is

$$\underset{a \in C}{\arg\max} \sum_{A_i \in Advisors} w_i \cdot s(A_i, a, C)$$

Strengths and Advisors' varying comment frequencies motivated new reinforcement learning algorithms (Petrovic and Epstein, 2008). In each of them, all Advisors begin with a uniform, small weight. Then, one training instance at a time, the weight of each Advisor whose set of the comments is judged appropriate is incremented, and that of each Advisor whose set of the comments is judged inappropriate is decremented. Each weight reflects only instances on which the Advisor commented. This learning is expected to require only a few experiences: 20 contests at a game, 20 trips in the same maze, or 30 CSPs from a class.

Learning which representations are appropriate not only enables FORR to emphasize them, but also enables it to disregard the inappropriate ones. A *benchmark Advisor* makes randomly many comments with random strengths. (Each benchmark represents a category of tier-3 Advisors, for example, variable-selection and value-selection Advisors in ACE.) Benchmark Advisors do not participate in decisions but they earn learned weights. After learning, any Advisor whose weight is lower than its respective benchmark is not consulted, and any descriptive referenced only by such Advisors is no longer computed.

Learning appropriate representations improves performance. For example, in one set of experiments, shown in Table 1, a *run* had ACE learn on 30 CSPs and then tested it on 50 more CSPs from the same phase-transition problem class. The first two lines compare ACE's performance after learning weights for 40 Advisors to the performance of the best individual Advisor among them. Each experiment

Table 1: Performance on 50 problems with 50 variables, domain size 10, density 0.38, and tightness 0.2.

	Search tree size	*Solved*
Testing performance		
Best individual heuristic	30,024.66	84.0%
Best single ACE run	8,559.66	98.0%
Learning performance		
ACE (DWL)	13,708.58	91.8%
ACE (RSWL)	13,111.44	95.2%
ACE (RSWL-*d*)	11,849.00	94.6%
ACE (RSWL-κ)	11,231.60	95.0%

with ACE had 10 runs. Originally, DWL, the weight-learning algorithm that produced the best single run in Table 1, considered digression size. More recent algorithms have retained DWL's testing performance and provided improved performance during learning. RSWL compares the strength of an Advisor's comment on an action to the strengths of its comments on other actions. Variations (RSWL-*d* and RSWL-κ) include an estimate of the difficulty of an individual training instance (Petrovic and Epstein, 2008).

Learning about representations

Learning about representations seeks to acquire descriptives' values (e.g., board positions or maze travel) and to appraise procedures that make use of those values. A fundamental premise here is that experience can support those goals. This section addresses issues in learning about representations.

Problem uniformity

To be consistently expert, a system should have experience throughout the space in which it searches. For game playing, that means learning against both expert and inexpert competitors (Epstein, 1994). The expert opponent provides a model to imitate, while the inexpert opponent broadens the learner's experience. Hoyle plays more reliably against opponents at all levels when it trains this way. Because Ariadne's trips all learn about the same maze, breadth of experience there comes with travel. In some sense, Hoyle and Ariadne each explore a single set of possibilities.

In constraint solving, however, each problem has its own space, and there is only ostensible control over problem uniformity. Classes of CSPs not based on real-world data are randomly generated according to parameters (MacIntyre et al., 1998). Nonetheless, for any given search algorithm, the difficulty of problems within such a class has been shown to have a heavy tail (Hulubei and O'Sullivan, 2005). This lack of uniformity readily misleads a learner. Recall that ACE only learns from solved problems. Early in learning on a class of CSPs, ACE may encounter a problem so easy that most any advice will do. Learning may award high weights to inappropriate Advisors based on that solution path, so that no subsequent problems of even reasonable difficulty are solved.

This issue motivated *full restart* in FORR, under which unsuccessful learning is terminated automatically, tier-3 weights are reinitialized, and learning begins afresh on new problems from the same class (Petrovic and Epstein, 2008). Without full restart, ACE did not always learn to solve small problems at the phase transition (30 variables, domain size 8, density 0.26, and tightness 0.34) within a 5000-node resource limit. Within a fixed number of full restarts, it learned on every run, and reduced its use of computational resources during learning by about 55%.

Too many representations

Too many tier-3 Advisors not only slow learning (while each one generates its advice) but also may prevent learning entirely. Initially in a difficult problem class, for example, the contention in the advice from ACE's equally-weighted Advisors in dual pairs may well prevent the solution of any problem. That in turn prevents any learning. In our experience, even 40 tier-3 Advisors can be too many.

The solution implemented in FORR is to select a new random subset of tier-3 Advisors for each problem during learning (Petrovic and Epstein, 2008). Even when the full set is heavily biased toward inappropriate representations, this method eventually finds a subset that solves some problem, and goes on to strengthen the weights of those Advisors repeatedly over time. Together, full restart and random subsets have substantially improved learning performance. On the problems of Table 1, individual decision time was reduced by 45%, and overall search time by 79%.

Using a new descriptive

As emphasized earlier, to be a representation a descriptive must support appropriate behavior. We offer a telling recent example from ACE. The problem is RLFAP scene 11, the most difficult of the radio-link frequency problems for

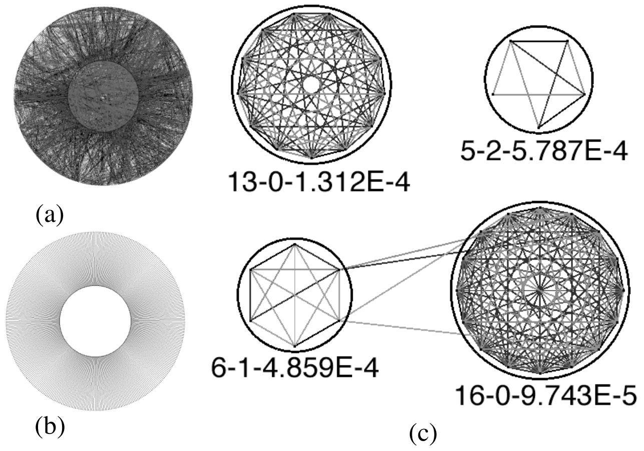

Figure 2: Descriptives for RLFAP scene 11. (a) Constraint graph with vertices on two concentric circles. (b) Constraints with tightness ≥ 0.3 form a bipartite graph. (c) Part of the cluster graph, whose darker edges also appear in (b).

Table 2: Performance on 50 structured problems with 200 variables. Time is CPU seconds per problem.

	Time	*Nodes*
Min Domain / Weighted Degree	83.580	12519.40
ACE with cluster Advisors	4.311	497.96

the assignment of broadcasting frequencies to radio towers in France (Cabon et al., 1999). Figure 2(a) shows its constraint graph on 680 variables.

Ideally, search begins with the most difficult parts of a problem. *Foretell* is an algorithm that predicts where they are likely to lie in a CSP before search begins (Epstein and Li, 2009). Tight constraints alone, such as those in Figure 2(b), may reveal little about a CSP's basic nature. On the other hand, disjoint *clusters* (dense, tight subproblems) can provide insight. *Foretell* detects clusters. Figure 2(c) is part of a *cluster graph* for the same problem. It shows 40 variables in 4 significant clusters found by *Foretell*, and includes all edges between them in the original graph.

A cluster graph is only a descriptive, however. Although it appears to convey meaningful data, it is not clear how an Advisor that prioritizes variables during search ought to use it. How should an Advisor select a variable within a cluster? Should a cluster-oriented Advisor concentrate on one cluster at a time, or hop from one to the next at the whim of other Advisors? If the former, in what order should clusters be addressed? Must a cluster be solved in its entirety before search moves on to the next one?

The right representation must answer these procedural questions. Selected manually, the right representation for a class of CSPs with non-random structure produced an order of magnitude speedup over the best individual CSP heuristic tested (Boussemart et al., 2004). (See Table 2.) For the problem in Figure 2, however, the best answers thus far have produced only a 12% speedup. Automatically learning the right answers to these questions is the subject of current research. A good descriptive alone is not enough.

Conclusions

In a complex domain, human experts typically use more than one representation. If a program is to solve problems there expertly, it too needs multiple representations. This requires both efficient storage structures and the ability to combine and apply them in a variety of productive ways. Representations have been described here in terms of descriptives and Advisors.

The FORR architecture provides a framework within which several programs learn to manage a portfolio of representations seamlessly. A set of Advisors is predicated, along with a set of descriptives. The most appropriate ones are then learned for a given problem class. After learning, low-weighted Advisors and descriptives unreferenced by high-weighted Advisors need no longer be computed. As a result, FORR-based programs display considerable performance improvement after learning.

Acknowledgements

Thanks to the FORR study group, particularly Smiljana Petrovic and Xingjian Li. ACE is a joint project with Eugene Freuder and Richard Wallace of the Cork Constraint Computation Centre. This work was supported in part by the National Science Foundation under awards IIS-0811437 and IIS-0739122.

References

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

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

Boussemart, F., F. Hemery, C. Lecoutre and L. Sais 2004. Boosting systematic search by weighting constraints. In *Proceedings of ECAI-2004*, 146-149. IOS Press.

Cabon, R., S. De Givry, L. Lobjois, T. Schiex and J. P. Warners 1999. Radio Link Frequency Assignment. *Constraints* 4: 79-89.

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. Toward an Ideal Trainer. *Machine Learning* 15(3): 251-277.

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

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

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

Epstein, S. L. and X. Li 2009. Cluster Graphs as Abstractions for Constraint Satisfaction Problems. In *Proceedings of SARA-09*, Lake Arrowhead, CA.

Hulubei, T. and B. O'Sullivan 2005. Search heuristics and heavy-tailed behavior. In *Proceedings of CP 2005*, 328-342. Berlin, Springer-Verlag.

MacIntyre, E., P. Prosser, B. Smith and T. Walsh 1998. Random Constraint Satisfaction: theory meets practice. In *Proceedings of CP-98*, 325-339. Springer Verlag.

Petrovic, S. and S. L. Epstein 2008. Tailoring a Mixture of Search Heuristics. *Constraint Programming Letters* 4: 15-38.

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.

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