Full Restart Speeds Learning

Smiljana Petrovic[1] **and Susan Epstein**[1,2]

[1]Department of Computer Science, The Graduate Center, The City University of New York, NY, USA
[2]Department of Computer Science, Hunter College of The City University of New York, NY, USA
spetrovic@gc.cuny.edu, susan.epstein@hunter.cuny.edu

Abstract

Because many real-world problems can be represented and solved as constraint satisfaction problems, the development of effective, efficient constraint solvers is important. A solver's success depends greatly upon the heuristics chosen to guide the process; some heuristics perform well on one class of problems, but are less successful on another. ACE is a constraint solver that learns to customize a mixture of heuristics to solve a class of problems. The work described here accelerates that learning by setting higher performance standards. ACE now recognizes when its current learning attempt is not promising, abandons the responsible training problems, and restarts the entire learning process. This paper describes how such full restart (of the learning process rather than of an individual problem) demands careful evaluation if it is to provide effective learning and robust testing performance.

Introduction

In the domain investigated here, an unsupervised learner searches for a solution to difficult combinatoric problems within a specified resource limit. If it fails on a problem, the learner tries to solve the next one. If it succeeds, the learner gleans training instances from its own (likely imperfect) trace, and uses them to refine its search algorithm before it continues to the next problem. Although a different sequence of decisions might have led to the learner's solution (or another, equally acceptable one) more quickly, the learner is restricted here to its own single search trace. The long-term goal of this work is to learn an effective, efficient combination of domain-specific heuristics that works well on a specific set of problems. Our thesis here is that *full restart* of the entire learning process can accelerate such convergence. Our principal result is the success of full restart on a given class of constraint satisfaction problems (*CSPs*), even though the learner is unsupervised and the quality of any individual solution must go unknown.

The program described here learns to solve CSPs. During search it uses a traditional CSP combination of heuristic decision making, propagation, and backtracking. Its learning algorithm refines the weights assigned to a large set of heuristics, based on feedback from the solver after each problem. Search and learning are closely coupled; weights are updated based on search results, and decisions made during search are based on the current weights. The program learned correctly without full restart, but under a high resource limit. This paper shows how, under proper control, even repeated full restarts can accelerate this learning without compromising performance.

This work is novel for several reasons. Previously, restart has been used either to make a fresh start on global search for solution to a single problem, or to make local search more resilient to local minima. In both cases, restart was meant to find a solution to a problem more quickly. The goal of full restart, however, is to accelerate the convergence of learning to a high-performance combination of heuristics, even though the optimal combination and the distance from it are unknown. Abandoning a learning process only to begin it again can be costly; we propose here a strategy that is both economical and effective.

Traditional restart reuses its training problems; some randomness varies the search experience on each new attempt. In contrast, full restart abandons a training problem entirely if no solution is found within a user-specified *step limit* (maximum number of search decisions), and substitutes another. Full restart also begins the entire learning process again under the direction of a user-specified *restart strategy* that compares the frequency and distribution of abandoned problems to a *full restart threshold*.

The success of full restart depends upon critical interactions among the step limit, the restart strategy, and the full restart threshold. The next section provides background on constraint satisfaction and describes the learning program. Subsequent sections motivate full restart, demonstrate its success, evaluate the influence of various parameters on learning performance, and discuss related work.

Constraint satisfaction problems

A *constraint satisfaction problem* (CSP) consists of a set of variables with associated domains and a set of constraints, expressed as relations over subsets of these variables. A *partial instantiation* of a CSP is an assignment of values to some proper subset of the variables. An instantiation is *consistent* if and only if all constraints over the variables in it are satisfied. Problems that may be expressed as CSPs include scheduling, satisfiability (SAT), graph-coloring, and Huffman-Clowes scene labeling.

A *solution* to a CSP is a consistent instantiation of all its variables. A *binary CSP* (all constraints are relations between two variables) can be represented as a *constraint graph*, where vertices correspond to the variables (labeled by their domains), and each edge indicates the existence of a constraint between its respective variables (labeled with consistent pairs of values). A *class* is a set of CSPs with the same parameters. For example, one way to define a class is with parameters <*n, m, d, t*>, where *n* is the number of variables, *m* the maximum domain size, *d* the density (fraction of edges out of *n(n-1)/2* possible edges) and *t* the *tightness* (fraction of possible value pairs that each constraint excludes) (Gomes, Fernandez, et al., 2004). Although CSPs in the same class are ostensibly similar, there is evidence that their difficultly may vary substantially for a given solution method (Hulubei and O'Sullivan, 2005). Our experiments here are limited to classes of solvable binary CSPs, generated randomly using methods and code shared by the CSP community.

Determining whether a CSP has a solution is an NP-complete problem; the worst-case cost is exponential in *n* for any known algorithm. Most instances of these problems can be solved with a cost much smaller than the worst-case cost, however.

The CSP search algorithm used here alternately selects a variable and then selects a value for it from its domain, incrementally extending a partial instantiation by a value assignment consistent with all previously-assigned values. When an inconsistency arises, search *backtracks:* the subtree rooted at the inconsistent node is pruned and another value from the domain of the same variable is tried. If every value in that variable's domain is inconsistent, the current partial instantiation cannot be extended to a solution, so the previously assigned variable in the instantiation is reassigned. Typically, extensive search occurs when many attempts lead to "near" solutions, and backtracking occurs deep in the search tree. Further pruning of the search tree is accomplished by some form of *propagation* to detect values that cannot be supported by the current instantiation. Here we use *MAC-3* to maintain arc consistency during search (Mackworth, 1977). MAC-3 temporarily removes currently unsupportable values to maintain *dynamic* domains for the variables. The size of a search tree depends upon the order in which values and variables are selected. Different *variable-ordering heuristics* (rules to select the next variable for instantiation) and *value-ordering heuristics* (rules to select the next value to be assigned to an already-selected variable) can support extensive early pruning and thereby speed search.

ACE, the Adaptive Constraint Engine

ACE (the Adaptive Constraint Engine) learns to customize a weighted mixture of heuristics for a given CSP class (Epstein, Freuder, et al., 2005). ACE is based on FORR, a problem-solving and learning architecture for the development of expertise from multiple heuristics (Epstein, 1994). ACE makes decisions by combining recommendations from procedures called *Advisors*, each of which implements a heuristic for taking, or not taking, an action. By solving instances of problems from a given class, ACE learns an approach tailored to that class. Advisors are organized into three tiers. Tier-1 Advisors are always correct. If a tier-1 Advisor comments positively, the action is executed; if it comments negatively, the action is eliminated from further consideration during that decision. The only tier-1 Advisor in use here is *Victory*, which recommends any value from the domain of the last unassigned variable. Tier-1 Advisors are consulted in a user-specified order. Tier-2 Advisors address subgoals; they are outside the scope of this paper. The decision making described here focuses on the heuristic Advisors in tier 3.

Each tier-3 Advisor can *comment* upon any number of actions; each comment has a *strength* which indicates the degree of support or opposition of the Advisor to the action. Each Advisor's heuristic view is based upon a descriptive property. An example of a value Advisor is *Max Product Domain Value*, which recommends values that maximize the product of the sizes of the dynamic domains of its neighbors. An example of a variable selection Advisor is *Min Dom/Deg*, which selects variables whose ratio of dynamic domain size to static degree in the constraint graph is a minimum. For each property, there are two Advisors, one that favors smaller values for the property and one that favors larger ones. For example, domain size is referenced by *Max Domain* and *Min Domain,* which recommend variables with the largest and smallest domains, respectively. Typically, one from each pair is known by CSP researchers to be a good heuristic, but ACE implements both of them, and has occasionally demonstrated that the other heuristic is successful for some problem classes. There are also two *benchmark Advisors*, one for value selection and one for variable selection, which do not participate in decisions but model random tier-3 advice. During testing, only an Advisor that has a weight larger than the weight of its respective benchmark is permitted to comment. When a decision is passed to tier 3, all its Advisors are consulted simultaneously, and a selection is made by *voting:* the action with the greatest sum of weighted strengths from all comments is executed.

During learning, after it solves a problem, ACE uses the *DWL* (Digression-based Weight Learning) algorithm to update its *weight profile,* the set of weights of its tier-3 Advisors. DWL learns from both *positive training instances* (decisions made along an error-free path extracted from a solution) and *negative training instances* (decisions leading to a *digression*, a failed subtree). DWL discards the decisions made within a digression. For each positive training instance, Advisors that *supported* it (included it among the Advisor's highest-strength preferences) are rewarded with a weight increment. Advisors that supported a negative training instance are penalized in proportion to the number of search nodes in the resultant digression. DWL also penalizes variable-ordering Advisors that supported selection of the last variable at the time of digression. DWL gauges the learner's developing skill across a sequence of learning problems. DWL weight increments depend upon the size of the search tree, relative to the best search so far (the minimal size of the search tree in all previous problems). Short solutions indicate a good variable order, so correct variable-ordering Advisors in a short solution will be highly rewarded. For value-ordering Advisors, short solutions are interpreted as an indication that that problem was relatively easy (i.e., any value selection would likely lead to a solution), and therefore result in only small weight increments for correct Advisors. A long search, in contrast, suggests that a problem was relatively difficult, so value-ordering Advisors that supported positive training instances there receive substantial weight increments.

The motivation for full restart

The motivation for full restart is to speed learning without compromising performance. Of necessity, ACE learns to solve CSPs with incomplete information. An omniscient teacher would require full knowledge of the exponentially-large search space for each problem to determine an optimal variable ordering and value ordering. Instead, ACE learns from its own solutions, and learns only after an instance is solved. This approach is problematic for several reasons. Recall that the number of decisions in a search tree measures ACE's overall performance, and is the basis for rewards and penalties. A particular Advisor, however, may be incorrect on some decisions that resulted in a large digression, and still be correct on many other decisions in the same problem. Moreover, a problem is solved only once, and learning is based on that solution. There is no guarantee that some other solution could not be found much faster, if even a single decision were different.

ACE is an effective learner. Min Dom/Deg (*mDD*, defined above), an "off-the-shelf," often-used heuristic, averaged 228.65 steps to find a first solution to a test set of 60 problems in <30, 8, 0.31, 0.34>. In contrast, ACE averaged 149.39 steps. This reduction is statistically significant at the 95% confidence level. ACE's performance is an average across 10 runs. In each run, ACE first learned weights for 36 heuristics on a new set of 30 problems and then tested those weights on the original set of 60 problems. The next challenge was to accelerate its learning.

Table 1: Post-learning performance of ACE, measured by average steps to solution, with and without restart. Without restart, reduced learning resources (the learning step limit) produce occasional unsatisfactory runs. With an appropriate restart strategy, however, learning resources can be reduced by an order of magnitude without compromising performance.

Class	<30, 8, 0.31, 0.34>			<30, 8, 0.18, 0.5>		
Restart strategy	None	None	4 out of 7 failed	None	None	4 out of 7 failed
Learning step limit	20000	2000	500	10000	1000	500
Run 1	145.13	145.12	144.47	71.80	**3324.42**	73.00
Run 2	149.17	150.10	147.85	71.07	71.38	72.37
Run 3	163.28	**6541.17**	163.98	69.72	69.72	71.95
Run 4	146.85	151.63	152.73	70.85	70.43	73.23
Run 5	153.25	**6373.50**	156.27	71.53	71.92	71.97
Run 6	144.30	144.02	154.63	71.43	72.43	75.82
Run 7	154.90	157.73	158.10	72.37	71.42	71.50
Run 8	150.27	154.55	153.25	69.75	73.87	72.43
Run 9	135.93	157.68	162.58	71.25	**3370.53**	72.78
Run 10	150.77	154.25	158.00	69.90	71.62	73.20

During search, a *step* selects either the next variable or a value for that variable, and the *step limit* is the maximum number of steps allowed to solve the problem. Although in most runs ACE was significantly better than mDD, limiting resources by lowering the step limit during learning produced occasional runs where ACE did not learn how to solve these problems well. This difficulty is illustrated in Table 1, which shows ACE's performance on 10 runs in two different classes. Inadequate runs appear in bold.

For difficult CSPs, the number of steps to solution with a given search algorithm may vary greatly from problem to problem in the same class, and can be modeled by a *heavy-tailed* distribution whose tail decreases according to a power-law (Gomes, Selman, et al., 2000). This variation is not necessarily due to the difficulty of an individual problem — on another, somehow randomized attempt, or with a different algorithm or with a different heuristic, the same problem may be solved quickly.

The extremely high average number of steps of the inadequate runs in Table 1 are not the effect of an algorithm occasionally approaching a difficult problem. On the inadequate runs, ACE consistently failed to solve problems that were successfully solved in other runs. (The same testing set was used in each run.) This indicates that DWL did not converge to an appropriate weight profile during learning. Figure 1 shows the weights of some selected Advisors after each of the first 5 runs for <30, 8, 0.31, 0.34>. ACE learns to discriminate between Advisors that minimize and maximize the same property; in all but runs 3 and 5, it assigned higher weights to the version corresponding to well-known, good heuristics. In contrast, the most highly-weighted heuristics from the inadequate runs never participated in testing during the other, successful runs (because of their worse-than-benchmark weights), and the highly-weighted heuristics from the other runs did not survive to participate in the testing in runs 3 and 5.

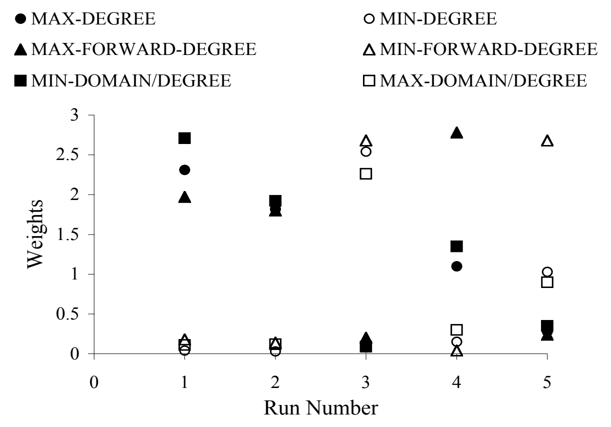

Figure 1: Weights learned for selected well-respected Advisors (solid shapes) and their opposites (hollow) in three adequate (runs 1, 2 and 4) and inadequate (runs 3 and 5) on problems in <30, 8, 0.31, 0.34>.

Efficient full restart

The incentive for decreasing the step limit is a extensive *learning effort*, measured here as total steps during learning. We show here that, on difficult problems, full restart with a lower step limit can eliminate occasional inadequate runs, and thereby maintain the high testing performance previously observed under a high step limit without full restart.

Recall that inadequate runs only arose when a higher standard (lower step limit) was set for the learner. Rather than extend training to more learning problems, with the hope that Advisor weights would eventually recover, we used full restart: abandon the learning process (and any weight profile) after some small number of unsolved instances, and start learning afresh on new problems.

Our model assumes that the responsibility for lengthy search is shared both by the inherent characteristics of the problem and by an inadequate weight profile. Inability to solve problems within the step limit reflects inadequacy in the weight profile. Indeed, an inadequate weight profile will fail to solve many problems, regardless of how high we set the step limit. With a good weight profile, however, most problems will be solved, even within a reasonably low step limit. Without full restart, the step limit is usually set high, so that the learner solves most of the problems and can learn from them.

Experimental design

For ACE, a *run* is a *learning phase* (a sequence of problems that it attempts to solve and uses to learn Advisor weights), followed by a *testing phase* (a sequence of fresh problems that it solves with learning turned off). All problems are from a single <*n, m, d, t*> class. Because problem classes are inherently of uneven difficulty, ACE's performance is evaluated over a set of 10 runs with different learning problems but the same testing problems. In the work reported here, ACE referenced 36 tier-3 Advisors during learning; during testing ACE included only those tier-3 Advisors whose weights exceeded their corresponding benchmarks. Experiments were performed on four classes of random, solvable, binary CSPs: <30, 8, 0.31, 0.34>, <30, 8, 0.315, 0.34> and the somewhat easier <30, 8, 0.16, 0.5> and <30, 8, 0.18, 0.5>.

Both problems and phases have termination conditions. The termination condition for testing was 10000 steps; for learning problems, the step limit was an experimental parameter that ranged from 200 to 1500. The termination condition for each testing phase in every run in all these experiments was the solution of the same 30 problems. We tried two termination strategies for the learning phase. In the *fixed-length* criterion, ACE was required to learn on specified number of problems before it began testing. Under full restart occurred, this count was reset to 0, so that ACE learned on at least the specified number of problems. The *expertise* criterion terminated the learning phase after ACE solved a specified number of consecutive problems.

Under full restart, after each problem ACE evaluated its learning progress with respect to a restart strategy. If, according to that strategy, learning was not progressing well, ACE performed a full restart: it discarded its learned weights and tackled a fresh set of learning problems. During learning, 20 full restarts were allowed; after 10 restarts, the step limit was increased by 50 on each full restart. This increment is intended to recover from an unreasonably demanding step limit. (A problem with *n* variables requires at least 2*n* steps.)

Our restart strategy was "full restart after *k* failed problems out of the last *m* problems." It allowed us to avoid full restart due to multiple but sporadic failures attributed to uneven problem difficulty rather than an inadequate weight profile. For the <30, 8, 0.31, 0.34> class, we tested values of *m* and *k* from 2 to 7; the best tested full restart strategy was "failure on 4 out of the last 7 problems." For an easy to satisfy strategy (e.g., failure on 2 out of 7) and a low step limit, frequent full restarts prevented ACE from learning on all but the easiest problems, and did not eliminate inadequate runs. A hard to satisfy strategy (e.g., failure on 7 out of 7), particularly with a high step limit, effectively prevented full restart and therefore still produced inadequate runs. The easier <30, 8, 0.16, 0.5> did better with a smaller number of failures to trigger restart ("2 out of 7" and "3 out of 7").

Full restart and the step limit

Full restart makes the step limit more important. Because ACE *fails* on a problem if it does not find a solution within the step limit, the step limit is the criterion for unsuccessful search. Because the full restart threshold directly depends upon the number of failures, the step limit becomes the performance standard for learning. Moreover, the step limit serves as a filter on problem difficulty: since ACE does not learn from unsolved problems, a lower step limit actually eliminates more difficult problems. An inadequate run, regardless of the step limit, has repeated failures from which the program cannot learn, and they consume considerable resources. Full restart should help the learner respond, early on, to learning that is not going well.

Figure 2 illustrates the relationship between search effort and the step limit. The circles there show the learning effort under a fixed-length criterion of 30. An extremely low step limit alone is insufficient to prevent inadequate runs. Frequent problem failures not only increase the number of full restarts (and hence the learning effort), but also leave the learner without training instances. Indeed, when presented with a 200-step limit, ACE required considerable resources and eventually began to increment the step limit (after 10 unsuccessful full restarts); only then could it learn.

With a relatively high step limit (e.g., 1000), many difficult problems are solved, and full restart triggers only on the rare occasions when a learning attempt is not promising. Nonetheless, every failure is expensive. With a low (but reasonable) step limit (e.g., 300), the learner fails on all the difficult problems, and even on some of medium difficulty, repeatedly triggering full restart until the weight profile is good enough to solve (almost) all the problems. A low step limit results in many failures, but relatively inexpensive ones.

This might suggest that reducing the step limit to some intermediate value would reduce learning effort (avoid overly expensive failures), but that further reduction of the step limit would incur overly frequent failures and increase learning effort. Nevertheless, as Figure 2 illustrates, learning with an intermediate step limit is more costly because it simply delays full restart and thereby forces learning from all but the most difficult problems. The learned weight profile is good enough so that failures are less frequent, and full restart is postponed. When full restart eventually triggers, the work on more problems (requiring relatively extensive effort) is abandoned.

Figure 3 chronicles performance in three sample runs with full restart. With a relatively high (1000) step limit, there were a few failures at the beginning of a run; ACE recovered without resorting to full restart, and was exposed to exactly 30 problems. With an intermediate (500) step limit, there were some full restarts at the beginning, and, once an acceptably good weight profile was learned, there were only sporadic failures; exposure was to 42 problems in all. With a relatively low (250) step limit, there were many failures at the beginning of each attempt at learning, and many full restarts before a good weight profile was established. Subsequent failures occurred, but were not frequent enough to trigger full restart; learning required 58 problems in all.

Achieving expertise

Under the fixed-length criterion and the step limits investigated here, full restarts occurred only early in a run. Once a good weight profile is established, there may be an occasional failure, but there are no full restarts. Consistent success in solving problems during learning indicates a good weight profile. Under the expertise termination criterion (solution of a specified number of consecutive problems), the number of learning problems is significantly reduced. For example, solving problems from <30, 8, 0.31, 0.34> with a 1000-step limit takes only an average of 13.10 problems when 10 consecutive solutions are required, instead of 32.40 under the fixed-length criterion when learning was on at least 30 problems. The reduction in learning effort is somewhat less dramatic (4323.58 steps for the entire phase instead of 8237.01 steps) because the expertise criterion excluded readily solved problems. With a lower step limit, failure is more frequent so that the expertise criterion is more demanding; only learning a good weight profile can provide consistent success under a low step limit. This increases the number of learning problems, but failure is inexpensive. With the step limit of 400, most problems provide training instances (i.e., are solved within the step limit), and, when full restart occurs, it is well warranted. With all step limits, testing performance was not compromised; it ranges from 143.06 to 153.63 steps. Figure 2 shows that with the expertise criterion, learning effort is consistently reduced compared to fixed-length criterion.

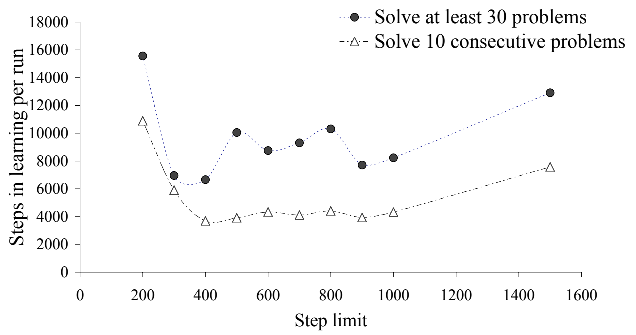

Figure 2: Average learning effort across 10 runs for two learning phase termination criteria, with full restart and different step limits on <30, 8, 0.31, 0.34>.

Discussion and future work

Restart on an individual problem is often effective. Randomized restart has successfully been applied to Internet traffic, scheduling (Sadeh-Koniecpol, Nakakuki, et al., 1997), theorem proving, circuit synthesis, planning, and hardware verification (Kautz, Horvitz, et al., 2002). On difficult CSPs, Rapid Randomized Restart effectively eliminates the heavy tail in the run time distribution of backtrack search on an individual problem (Gomes, et al., 2000). Just as traditional restart relies on fortuitous assignments, full restart relies on a fortuitous training set.

There are many ways to determine the appropriate restart cutoff value for an individual problem. If the runtime distribution of a problem is known, it is possible to compute an optimal fixed cutoff value; if the distribution is unknown, there is a universal strategy provably within a log factor of optimal (Luby, Sinclair, et al., 1993). Another successful strategy increases the cutoff value geometrically (Walsh, 1999). When even partial knowledge of the effort distribution is known, and data on the search process is available, an appropriate restart cutoff can be dynamically

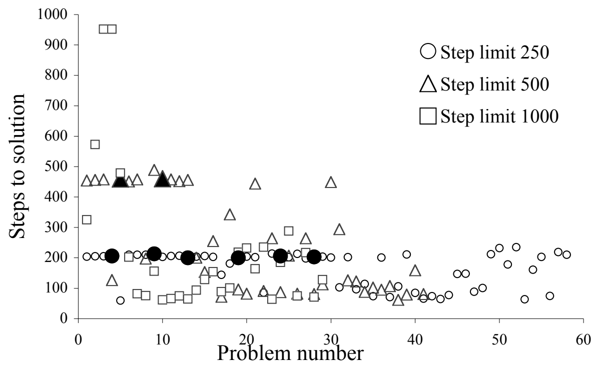

Figure 3: Steps and full restarts during learning on at least 30 problems in <30, 8, 0.31, 0.34> under different step limits. Solid shapes flag restarts. There were no restarts with a 1000-step limit.

determined (Kautz, et al., 2002). Still another approach dynamically identifies problem features that underlie runtime distribution. It uses them to partition problem instances into classes with smaller runtime variability, with a different cutoff for each subset (Ruan, Horvitz, et al., 2003).

Further improvements with traditional restart have been achieved with different implementations of randomness. Other research has changed the return point in the search tree (Zhang, 2002), or changed the temperature in simulated annealing (Sadeh-Koniecpol, et al., 1997). Probing (restart with a cutoff value of one backtrack) has been used to test the ability of a heuristic to remain on a solution path in very small problems (Beck, Prosser, et al., 2003).

Potentially, repeated full restarts could demand many problems from a single class. ACE has libraries with as many as 10000 problems in each class, but producing enough problems may be impractical in some cases. Instead, we intend to reuse unsolved problems by introducing randomness or perturbing them after full restart. Previous experience with problems and their difficulty should also allow us to implement boosting with little additional effort during learning (Schapire, 1990).

We plan further automation of the full restart mechanism. Ultimately for a specific class of problems, ACE should learn the appropriate step limit and any expertise parameters on its own. Analysis will also be extended to other kinds of CSPs and to mixtures of solvable and unsolvable problems.

Conclusion

Learning systems with limited feedback and limited resources are prone to occasional unsatisfactory performance. We showed here that one such system responds well to full restart. ACE receives very limited feedback from its attempts to solve constraint problems. Because of the intrinsic nature of constraint solving, there is no supervisory commentary on individual decisions, and the delayed rewards cannot truly reflect success during training.

As reported here, full restart can speed learning and improve robustness without sacrificing performance. Without full restart, curtailing resources may result in inadequate learning. With full restart, however, learning can be robust, even when resources are reduced by an order of magnitude. Full restart has proved most effective when it responds to the frequency of recent problem failure and when learning terminates after some number of consecutive problems has been solved.

Acknowledgments

This work was supported in part by NSF IIS-0328743 and PSC-CUNY. We thank the ACE research group, Richard Wallace, and the anonymous referees for their thoughtful suggestions and comments.

References

Beck, C., P. Prosser and R. J. Wallace (2003). Toward Understanding Variable Ordering Heuristics for Constraint Satisfaction Problems. *Fourteenth Irish Artificial Intelligence and Cognitive Science Conference - AICS 2003*, pp. 11-16.

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

Epstein, S. L., E. C. Freuder and R. Wallace (2005). Learning to Support Constraint Programmers. *Computational Intelligence* 21(4): 337-371.

Gomes, C., C. Fernandez, B. Selman and C. Bessiere (2004). Statistical Regimes Across Constrainedness Regions. *Principles and Practice of Constraint Programming (CP-04)*, pp. 32-46, Springer, Toronto, Canada.

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

Hulubei, T. and B. O'Sullivan (2005). Search Heuristics and Heavy-Tailed Behavior. *Principles and Practice of Constraint Programming (CP-05)*, pp. 328-342, Berlin: Springer-Verlag.

Kautz, H., E. Horvitz, Y. Ruan, C. Gomes and B. Selman (2002). Dynamic restart policies. *Eighteenth National Conference on Artificial Intelligence*, pp. 674 - 681, AAAI Press, Edmonton, Alberta, Canada.

Luby, M., A. Sinclair and D. Zuckerman (1993). Optimal Speedup of Las Vegas Algorithms. *Israel Symposium on Theoretical Aspects of Computer Science ISTCS*, pp. 128-133.

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

Ruan, Y., E. Horvitz and H. Kautz (2003). Hardness-Aware Restart Policies. *IJCAI-03 Workshop on Stochastic Search Algorithms*, Acapulco, Mexico.

Sadeh-Koniecpol, N., Y. Nakakuki and S. R. Thangiah (1997). Learning to Recognize (Un)Promising Simulated Annealing Runs: Efficient Search Procedures for Job Shop Scheduling and Vehicle Routing. *Annals of Operations Research* 75: 189-208.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning* 5(2): 197--227.

Walsh, T. (1999). Search in a small world. *IJCAI-99*, pp. 1172--1177, Morgan Kaufmann Publishers, San Francisco, CA, Stockholm, Sweden.

Zhang, H. (2002). A random jump strategy for combinatorial search. *Sixth International Symposium on Artificial Intelligence and Mathematics*, pp. 9, Fort Lauderdale, FL.