Category: adaptive behavior

Evaluation Criteria for Genetically-Tuned Problem-Solving Experts

David Sturgill
Computer Science Dept.
Baylor University
PO Box 97356 Waco, TX 76798-7356
sturgill@cs.baylor.edu
254 710 - 3876

Gautam Pant
Computer Science Dept.
Baylor University
PO Box 97356 Waco, TX 76798-7356
pant@cs.baylor.edu
254 710 - 3876

Abstract

Genetic algorithms have proven to be a powerful tool in solving computationally difficult problems. We present a technique for using genetic algorithms to learn a domain-specific problem solver. This technique uses a collection of search-based problem solvers, each using a different, genetically biased search procedure. Problem solvers compete in parallel to show that their search procedure, with its genetic bias, is the best; faster systems are permitted to pass their genetics on to the next generation. We compare two different techniques for learning based on two basic policies for evaluating the fitness of a problem solver. One policy assumes that problem solvers should be equally good at all problems. The alternative fitness estimate assumes that a search procedure may be very good at one kind of problem, but very bad at other kinds of problems. This paper evaluates the effectiveness of this learning technique and compares the two different learning policies in the domain of nonlinear planning.

1 Introduction

For many computationally hard problems, search is the only completely reliable means of obtaining a solution. Weak search techniques have the advantage of being domain-general but often suffer from a combinatorial explosion in the state space. Strong techniques can reduce search by exploiting the structure of the domain, but successful application of strong techniques to one class of problems may not help with different problems. Learning offers the possibility of acquiring the advantages of strong techniques through domain-general means.

This paper presents a technique for using genetic algorithms to learn how to solve search problems more quickly. Effectively, we learn a good problem solver for some domain of interest. We compare two different policies for evaluating the fitness of a search-based problem solver. One policy is built around the assumption that a good problem solver should perform well on all problems in the domain of interest. The alternative policy reflects the belief that, even in a single domain, a good problem solver will be better at some problems than it is at others.

In the following sections, we first describe our technique for using genetic algorithms to learn more effective search procedures. We describe two alternative policies for measuring the fitness of a search-based problem solver. We then compare our approach to related work in applying genetic algorithms to computationally difficult problems. We describe an implementation of our technique in the domain of nonlinear planning and present empirical results against a suite of simple planning problems. Finally, we summarize the most significant results of this work and consider some open issues that remain to be explored and directions for future research.

2 Learning to be a Better Problem Solver

We separate the task of learning from the task of problem solving. Figure 1 illustrates the two-level organization of our approach. At the *domain level*, domain-specific search is used to solve some problem of interest. At this level, a pool of different search-based systems compete in parallel to solve the problem; each system uses a different search procedure to solve the same problem. The intent is that problem solvers be sufficiently different that one will solve the problem quickly. Individual systems are differentiated by their genetic makeup (their *chromosome*). The search pro-

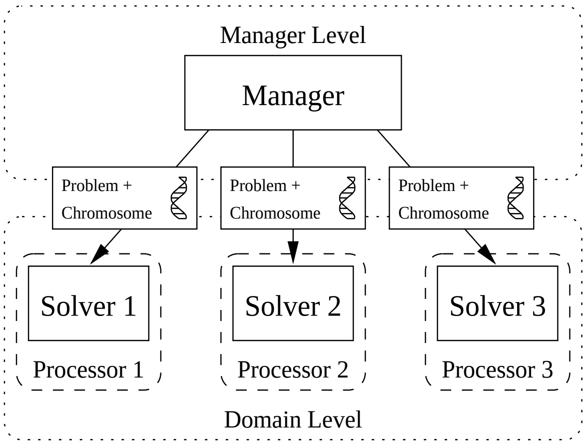

Figure 1: Two-level organization of problem-solving and learning tasks

cedure for the problem solvers at the domain level is sensitive to a number of tunable parameters specified in the chromosome. The problem solver is designed so that adjusting these parameters may have a substantial effect on the search order, but it will not affect search completeness. Learning is a process of refining the genetic bias of a problem solver in an effort to reduce search and improve performance.

Learning is handled at the *manager level*. The manager is responsible for selecting a genetic bias for each problem solver. As the system is exposed to more problems, it attempts to refine the genetic makeup of its pool of problem solvers. When a new problem must be solved, the manager gives each problem solver a copy of the problem along with a unique chromosome. All problem solvers attempt to complete the problem using their assigned chromosome. Problem solvers work in parallel, effectively trying to show that their chromosome is better than those of their competitors. As soon as one finds a solution, it is reported to the user via the manager. However, the manager permits the remaining solvers to continue to work on the problem until they find a solution or a new problem is submitted. The solution time for each solver gives the manager a measure of the fitness for the corresponding chromosomes. Periodically, the manager runs a simple genetic algorithm[Goldberg, 1989] to produce a new population of chromosomes.

This two-level organization demonstrates two different kinds of search occurring concurrently. At the domain level, problem solvers search through the space of possible solutions to the target problem. Meanwhile, the manager performs a search through the space of possible problem solvers. This separation offers a number of attractive properties.

- As the system is exposed to more problems, its performance can be expected to improve. Ideally, the system will learn a good way to solve problems like the ones it has seen in the past. Thus, even in a single domain, it may learn a bias that reflects the bias in the kinds of problems that actually occur.

- Although the problem solvers are domain-specific, the manager is completely domain-general. It refines the genetic makeup of its problem solvers without any dependence on the operation of the solvers or the meaning of their chromosomes.

- Learning does not affect the quality of the solutions generated by the problem solvers. Solvers are implemented so that their genetic bias influences search order but not search completeness. Thus, even if the genetic algorithm learns a bad problem solver, it will never produce an incorrect solution.

3 Related Work

Genetic algorithms have proven to be an effective tool at solving computationally difficult problems. Perhaps the most common approach is to use genetic algorithms directly to develop a solution to some problem of interest. Each chromosome in the population can be interpreted as an approximate solution. An appropriate domain-specific fitness function can be used to discriminate between better and worse approximations of a solution. This approach has been used effectively in domains like boolean satisfiability [De Jong and Spears, 1989], scheduling [Davis, 1985] and the traveling salesperson problem [Michalewicz, 1992, Jog et al., 1989, Grefenstette, 1987].

Our approach differs from these techniques in that genetic algorithms are used to synthesize a problem solver rather than a problem solution. While the chromosome population used by direct methods consists of approximately correct solutions, the problem solvers used by our technique are guaranteed to be correct but exhibit approximately optimal performance. Since it is possible to tell when an exact solution has been found, direct methods generally enjoy obvious termination conditions. Since it may be unclear whether or not an optimal problem solver even exists, appropriate termination conditions for our learning procedure are much less obvious. This lack of a termination condition is compensated by an obvious, domain independent basis for evaluating fitness. Since improved solution time is always the desired outcome, no separate fitness function must be developed.

There is a sense in which our learning technique is a very restricted example of genetic programming [Koza, 1992]. Both techniques seek to derive a program or program fragment that accomplishes a specific task. Our approach focuses on learning only an effective search order rather than learning an entire problem solver. Restricting the scope of learning means that our technique cannot explore the space of all possible problem solvers. However, this restriction permits guarantees about the correctness of the resulting problem solvers that would not be possible if all features of the problem solver were a product of genetic programming.

Our policy of refining the performance of a search-based system through parameter adjustment is a common technique and is reminiscent of such well-known systems as Samuel's checkers player [Samuel, 1967]. The technique described in this paper takes advantage of the fact that a small change in the search procedure can have a dramatic influence on search performance. This phenomenon has been effectively exploited as a basis for competitive parallel search techniques [Ertel, 1990, Hogg and Williams, 1993, Sturgill and Segre, 1997]. If different search procedures exhibit sufficiently varied performance, it may be possible to reduce expected solution time by simply performing multiple, different searches in parallel. As soon as one one of the searches finds a solution, work done by the others can be discarded. The parallel operation of problem solvers at the domain level is intended to exploit this variation in performance across different search procedures. This variation is also the basis for our learning mechanism. The hope is that small refinements in the search ordering policy may result in substantial reductions in solution time.

4 Fitness and Search

Search problems represent a special situation in which to try to learn. While some search-based systems may be uniformly better than others, there are generally no uniformly best systems. For every problem instance, there is a search order that is best. Even a good solver will be better on some problem instances than it is on others. Measuring fitness of a chromosome based on its performance on a single problem instance might give a poor picture of its overall quality. To compensate, the manager uses performance of a chromosome across a number of problem instances as the basis for its fitness.

This variation in performance across problem instances also obscures the question of what kind of problem solver should be learned; is it more appropriate to learn a system that solves many problems quickly or is it better to learn systems that perform well on individual problem instances? We address this question by comparing two contrasting policies for learning.

- **Generalist:** Under the generalist policy, a chromosome is assigned a fitness value based on its total solution time across 10 problems. The intent is that this will foster problem solvers that are proficient on a general population of problems. Since different problems vary in difficulty, summing solution times across multiple problem instances would give disproportionate significance to difficult and easy instances. To give each problem instance equal weight in the fitness, solution time for a problem instance is normalized with respect to the average solution time for that problem across the rest of the chromosome population. Thus, in a population of n chromosomes, if $t_{i,j}$ is the reciprocal of the solution time [1] for chromosome i on problem instance j, the fitness for chromosome k is given by:

$$\mathit{fitness}_k = \sum_{j=1}^{10} \frac{n\ t_{k,j}}{\sum_{i=1}^{n} t_{i,j}}$$

- **Specialist:** Under the specialist policy, chromosomes are considered good as long as they solve some problems quickly. Since the only basis of comparison is their collection of peer chromosomes, performance is considered good if it surpasses that of peer solvers. Under the specialist policy, solution times are normalized just as they are in the generalist case. However, the chromosome k is assigned a fitness value according to its maximum, normalized solution time.

$$\mathit{fitness}_k = \max_{j=1}^{10} \frac{n\ t_{k,j}}{\sum_{i=1}^{n} t_{i,j}}$$

Informally, the specialist policy is intended to encourage the development of a community of expert problem solvers, each specialized to solve a particular class of problem instances. Of course, an ideal problem solver

[1] Fitness is based on reciprocal solution time so that greater fitness values will correspond to better performance.

would be good at all different types of problem instances. However, the two-level learning system described in this paper will perform well even if its pool of problem solvers includes only a few that are suited to the current problem. When a problem is submitted to the manager, all problem solvers compete in parallel to solve it. The *turnaround time*, the time it takes for the manager to respond with a solution, reflects the performance of the fastest solver working on the problem. Turnaround time will be low as long as each generation contains at least one chromosome that is well-suited to a given problem. We believe that the specialist learning policy has the potential to reduce turnaround time by allowing chromosomes to specialize on different types of problems.

5 Evaluation

To evaluate the proposed learning technique and to compare the performance of the alternative fitness metrics, we have implemented a manager and a search-based problem solver for STRIPS-style planning. A *plan* is a sequence of *operators* that transforms a given initial state to some desired goal state. States are represented as a set of propositions that are true in the state. Each operator is defined by a *precondition list* (propositions that must be true in order to apply the operator), an *add list* (propositions that become true after the operator is applied) and a *delete list* (propositions that are no longer true after the operator is applied). Our problem solver is modeled after the systematic, nonlinear planner [McAllester and Rosenblitt, 1991]. Here, a plan is defined as a partially ordered set of operators rather than a totally ordered sequence. The partial order among operators represents two kinds of ordering constraints:

- A *causal link* from operator a to operator b indicates that a must be applied before b because a is used to satisfy a precondition π of b.
- If there is a causal link from a to b, and some operator c has π in its delete list, operator c is called a *threat*. If c is applied between a and b, the preconditions of b may no longer be true when it's time to apply it. A *safety condition* is an ordering constraint that requires operator c to occur either before a or after b.

Rather than using propositions alone, our planner uses a lifted representation in which preconditions, add lists and delete lists may contain variables.

To emable learning, our planner is supplemented with many search parameters that are specified in a 120-bit chromosome. The value of this chromosome influences the search order in two ways. Conjunctive search order is determined by associating a score with each unsatisfied precondition and each potential threat. The search procedure gives priority to working on those parts of the plan that have the best score. This score is a weighted sum based on the answers to the following:

- In general, is it better to add causal links or safety conditions first?
- For an unsatisfied precondition:
  - Does the precondition require addition of a new operator to the plan?
  - How many different operators are capable of satisfying the precondition?
  - How many other operators in the plan have identical preconditions?
  - How many operators in the plan are capable of deleting this precondition?
  - Is the precondition ground or does it contain variables?
- For a potential threat, is the threatened precondition ground or does it include variables?

The weighting associated with each of these features is determined by the chromosome.

Our planner uses a depth-first iterative deepening search procedure. It begins by considering only very small plans. If no small plans are found, it considers larger and larger plans. Thus, the set of plans it considers early in the search is determined by its metric for plan size. This metric is computed as a weighted sum of the number of operators, the number of unsatisfied preconditions and the number of potential threats. Again, the significance of each of these features is determined by the chromosome.

We generated a pool of 50 planning problems. All 50 were examples of sliding tile puzzles like the one in Figure 2. A solution for one of these problems was a plan for moving a particular tile to a specified destination. The suite of problems was randomly divided into 25 training problems and 25 testing problems. The system was allowed to learn for 45 generations on the training problems using a population of 100 chromosomes. Performance of these chromosomes was then evaluated using the 25 testing problems. This separation between training and testing reflects an intent that the system learn more than just shortcuts for solving the 25 training problems quickly. Ideally, the

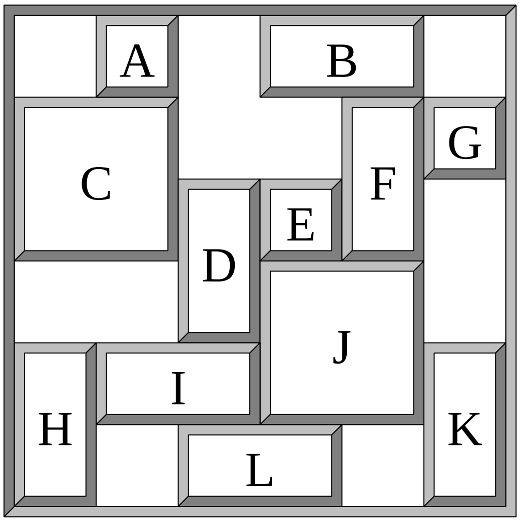

Figure 2: Sample problem for planning domain.

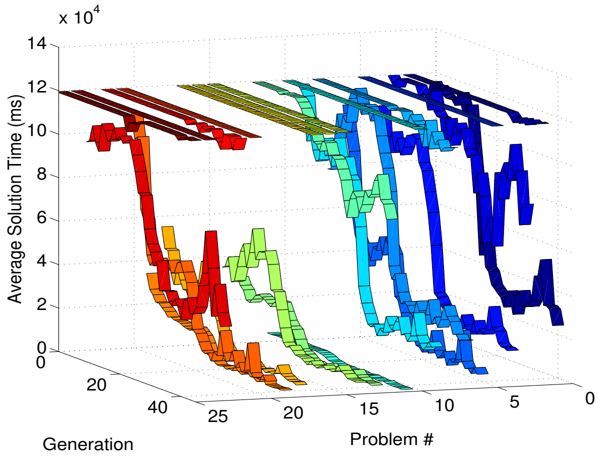

Figure 3: Average solution time over 45 generations of the generalist learning policy

learned search bias will improve performance on the 25 problems in the testing set even though the system is not permitted to learn from these.

Tests were conducted on a network of workstations running the Linux operating system. Solvers were run on a collection of 9 75Mhz Pentium systems and the manager was run on a 166Mhz Pentium. Since the manager's population of 100 chromosomes is much larger than the number of available processors, each processor was used to run many problem solvers in sequence.

Problems in both the training and testing sets are quite varied in difficulty. To prevent a solver from spending most of its time on a small number of very difficult problems, search time was limited to two minutes for each problem. If a search exceeded this bound, the problem went unsolved and the threshold value of 120 seconds was optimistically taken as the solution time for the corresponding chromosome.

Figure 3 presents performance of the planner using the generalist learning policy. Each line in the figure plots the solution time for one of the 25 testing problems. The vertical axis gives the average solution time in milliseconds over all 100 chromosomes in each generation. The generation axis presents the performance change over 45 generations of learning. Observe that, as lines move in the direction of higher generation numbers, they tend to drop with respect to average solution time. This demonstrates how experience on the training problems generalizes to the improved performance on many of the testing problems.

Seven of the 25 testing problems in Figure 3 are never solved within the two-minute time limit. These exhibit no visible performance improvement, but since their actual solution time is unknown, the effectiveness of learning on these problems is also unknown. All of the remaining problems show some performance improvement with experience. In fact, some problems that are unsolved in the first generation become solvable with sufficient experience.

Figure 4 presents solution times under the specialist learning policy. This learning system eventually solves the same 18 problems as those solved within two minutes by the generalist system. Although the influence of learning appears to be less smooth here, most problems show significant performance improvement after 45 generations. Some even show more than a 10-fold reduction in solution time.

Figure 5 compares summary results for learning under both the generalist and specialist policies. Each line shows average solution time for the 18 problems that are solved within the time bound. Thus, a line in this figure represents the average of many lines from Figure 3 or Figure 4. The figure indicates that the specialist policy does not yield as much performance improvement as the generalist policy. Although specialist chromosomes perform better in some regions of the graph, the generalist policy exhibits a distinct advantage after the twentieth generation. It should be noted that this figure gives disproportionate weight to the more difficult problems. Performance improvement on a difficult problem will have a greater influence on the total solution time than equivalent performance improvement on an easy problem. However, even when solution times are normalized to give equal emphasis

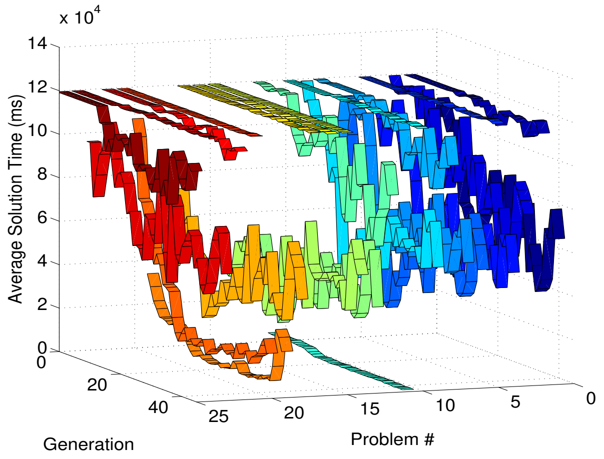

Figure 4: Average solution time over 45 generations of the specialist learning policy

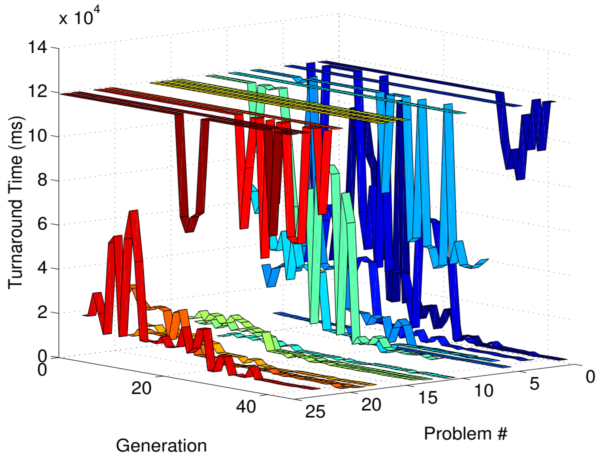

Figure 6: Turnaround time over 45 generations of the generalist learning policy

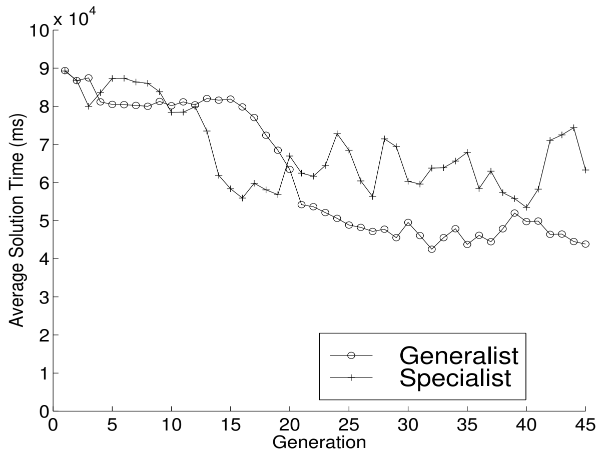

Figure 5: Comparison of average chromosome performance in generalist and specialist learning policies

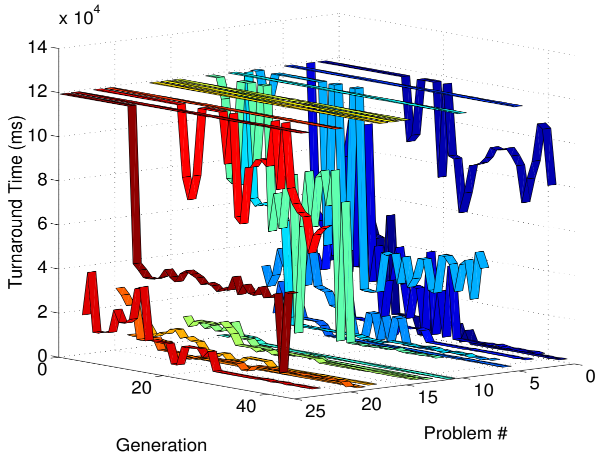

Figure 7: Turnaround time over 45 generations of the specialist learning policy

to all problems, the generalist policy demonstrates a similar advantage.

By presenting the average solution times at each generation, Figures 3 and 4 focus on the general fitness of the chromosome population in each generation. This gives a measure of how much the system is learning, but it may not be the most appropriate measure of overall performance improvement. Since performance on an individual problem depends on turnaround time, the solution time of the fastest chromosome on each problem may be a more meaningful measure of learning. Figure 6 plots the performance of the fastest chromosome on each problem using the generalist learning policy. Figure 7 gives a similar plot for learning with the specialist policy. Although these figures show that turnaround times are typically much lower than average solution time, there is still visible performance improvement in most problems.

Figure 8 gives summary results for Figures 6 and 7. This figure plots turnaround time under each polciy averaged across all 18 solved problems. Observe that, while the generalist policy showed an advantage in Figure 5, the specialist policy seems to be the better system here. Thus, while the generalist policy seems to produce a chromosome population that is more fit on average, the population under specialist learning seems to contain chromosomes that are better suited to individual problems.

The higher degree of variation in chromosomes learned under the specialist policy is demonstrated in Figure 9. Each point in this figure represents one of the 18 problems solved during the testing phase. This figure plots

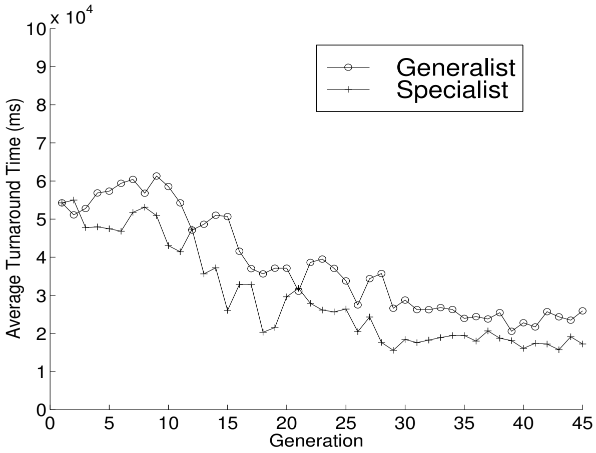

Figure 8: Comparison of turnaround time in generalist and specialist learning policies

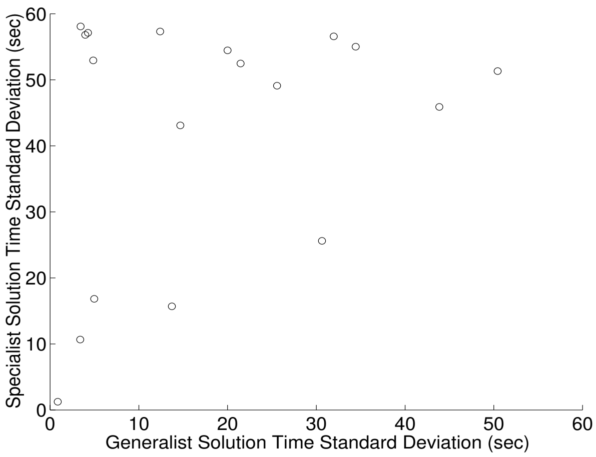

Figure 9: Comparison between standard deviation in solution times for specialist and generalist learning policies after 45 generations of learning

the performance on each problem after 45 generations of learning. However, instead of plotting actual solution times, the figure gives the standard deviation in solution time across all 100 chromosomes. Small values indicate a high degree of similarity in the population, while large values indicate greater variation in performance from one chromosome to the next. The X axis plots the standard deviation in solution times for chromosomes learned under the generalist policy. The Y axis gives corresponding values for the specialist policy. Observe that nearly all points in this graph lie above the diagonal, indicating that chromosomes in the specialist population exhibit a higher degree of performance variation than those in the generalist population.

6 Conclusion

We find many results of this work to be encouraging. Firstly, the search bias learned against a suite of training problems seems to generalize to improved performance on similar problems outside the training set. We take this as evidence that the system is learning general properties of the target domain rather than simply learning to solve a particular set of problems quickly. Secondly, learning resulted in performance improvement in the best performance of chromosomes in each generation and not just the average performance. This indicates our technique was able to develop new, better chromosomes, and did not not simply filter out the poor ones as it learned. Finally, the specialist learning policy seems to be effective at developing specialized chromosomes for particular kinds of problems. Although the overall fitness of chromosomes learned under the specialist policy is not as high as the generalist policy, specialist seems better at producing problem-solving experts.

We recognize one particular weakness with our current planner implementation. The influence of the chromosome on the search order is rather restricted. Since the chromosome may influence search order only by adjusting the weights on hand-selected search parameters, the potential for learning is limited to the optimal adjustment of these parameters. We have begun to experiment with more general mechanisms for introducing a genetic bias into the search procedure. Although this gives a substantial increase in the chromosome size, we believe that it offers greater potential to tailor system performance to a particular domain or class of problems.

As work on this parallel learning and problem solving technique continues, we are implementing problem solvers for additional domains (*e.g.* TSP). We are also investigating alternative learning policies that vary the order in which training problems are presented and the order in which different parts of the chromosome are learned. We hope that the positive results on planning will carry over to new domains and that refinement of the learning mechanism will capture greater potential for performance improvement.

References

[Davis, 1985] Davis, L. (1985). Job shop scheduling with genetic algorithms. In Grefenstette, J. J., editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications.*

[De Jong and Spears, 1989] De Jong, K. A. and Spears, W. M. (1989). Using genetic algorithms to solve np-complete problems. In *Proceedings of the Third International Conference on Genetic Algorithms*.

[Ertel, 1990] Ertel, W. (1990). Random competition: A simple but efficient method for parallelizing inference systems. In *Parallelization in Inference Systems*, pages 195–209. Springer.

[Goldberg, 1989] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

[Grefenstette, 1987] Grefenstette, J. J. (1987). Incorporating problem specific knowledge into genetic algorithms. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann.

[Hogg and Williams, 1993] Hogg, T. and Williams, C. P. (1993). Solving the really hard problems with cooperative search. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 231–236. AAAI Press/MIT Press.

[Jog et al., 1989] Jog, P., Suh, J. Y., and Gucht, D. V. (1989). The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the travelling salesman problem. In *Proceedings of the Third International Conference on Genetic Algorithms*.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming*. The MIT Press.

[McAllester and Rosenblitt, 1991] McAllester, D. and Rosenblitt, D. (1991). Systematic nonlinear planning. In *AAAI-91*.

[Michalewicz, 1992] Michalewicz, Z., editor (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

[Samuel, 1967] Samuel, A. (1967). Some studies in machine learning using the game of checkers II. *IBM Journal of Research and Development*.

[Sturgill and Segre, 1997] Sturgill, D. and Segre, A. M. (1997). Nagging: A distributed, adversarial search-pruning technique applied to first-order logic. *Journal of Automated Reasoning*.