Proceedings of the ML-93 Workshop on Knowledge Compilation and Speedup Learning, Amherst, MA, June 1993.

Knowledge Compilation and Speedup Learning in Continuous Task Domains

Juan Carlos Santamaría
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
carlos@cc.gatech.edu

Ashwin Ram
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
ashwin@cc.gatech.edu

Abstract

Many techniques for speedup learning and knowledge compilation focus on the learning and optimization of macro-operators or control rules in task domains that can be characterized using a problem-space search paradigm. However, such a characterization does not fit well the class of task domains in which the problem solver is required to perform in a continuous manner. For example, in many robotic domains, the problem solver is required to monitor real-valued perceptual inputs and vary its motor control parameters in a continuous, on-line manner to successfully accomplish its task. In such domains, discrete symbolic states and operators are difficult to define. To improve its performance in continuous problem domains, a problem solver must learn, modify, and use "continuous operators" that continuously map input sensory information to appropriate control outputs. Additionally, the problem solver must learn the contexts in which those continuous operators are applicable. We propose a learning method that can compile sensorimotor experiences into continuous operators, which can then be used to improve performance of the problem solver. The method speeds up the task performance as well as results in improvements in the quality of the resulting solutions. The method is implemented in a robotic navigation system, which is evaluated through extensive experimentation.

1 INTRODUCTION

Speedup learning deals with the issue of improving the performance of a problem solver with its experience. Most of the techniques that have been developed for speedup learning fall in two broad categories: learning new "macro-operators" by composing sequences of original operators (e.g. Fikes, Hart, & Nilsson, 1972; Mitchell, Keller, & Kedar-Cabelli, 1986; DeJong & Mooney, 1986), and learning some form of control knowledge that can be used to select which operator to try next (e.g. Mitchell, Utgoff, & Banerji, 1983; Laird, Rosenbloom, & Newell, 1986; Minton, 1990). These techniques were designed under the assumption that the problem solver conducts some form of problem-space search (e.g., applying operators to transform the current state into the goal state). However there are some domains in which the standard problem-space search paradigm does not fit well. This in turn is problematic for the applicability of the techniques described above. In particular, consider the class of problem domains in which the problem solver requires continuous performance in order to accomplish its task. We call these *continuous* domains.

There are many continuous domains and tasks in the real world. For example, consider driving a car on a highway. The problem solver must continuously decide how much pressure to apply to the gas to accelerate the car while entering the highway, how and when to control the steering of the car to change lines without colliding with other cars, how to decelerate the car while exiting the highway, etc. The problem solver is required to monitor different variables in its environment and to control several parameters in an on-line, continuous manner to successfully accomplish its tasks.

The standard problem-space search paradigm does not apply well to continuous domains since discrete and explicit states and operators are difficult to define. In the above example, the positions and velocities of the car (the "states"), and the actions of the driver such as the pressure on the gas pedal (the "operators"), are real-valued functions that continuously vary with time. In this paper, we address the issue of speedup learning and knowledge compilation for a problem solver performing in continuous task domains. We claim that in order to improve the performance of a continuous task in a continuous problem domain, a problem solver must be able to learn, modify, and use "continuous operators" that map input sensory information to control outputs. The difference between continuous operators and traditional STRIPS-like operators is that the former provide continuous control outputs ("results") based on continuous sensory information or, more generally, based on continuous representations of the operators' "preconditions".

One problem with continuous operators is that they are very difficult to design precisely by hand, especially if accurate models of the problem solver and of the environment are not known in advance. It is desirable for the problem solver

to be able construct and refine continuous operators automatically, for example, by first using general guidelines to accomplish the task and then using the resulting experiential knowledge to build continuous operators. Speedup learning in continuous domains can then be achieved by grouping continuous operators together into what may be thought of as "continuous macro-operators", and by learning heuristic applicability conditions for continuous operators and macro-operators. Knowledge compilation, then, can be thought of as the process of learning, through experience, operationalized knowledge about the environmental conditions (or sensory inputs) that should be associated with particular control outputs, while at the same time grouping sequences of sensory inputs and associated control outputs into larger chunks that can guide performance in future situations.[1]

Thus, a continuous problem-solving system operating in continuous domains accomplishes speedup learning when it can, through experience, begin to directly apply continuous operators that were successful in the past without going through the search process of finding useful control outputs for the current environmental situation. Such a system system also accomplishes knowledge compilation when it can compile its sensorimotor experiences into relevant continuous operators and macro-operators and, at the same time, find the conditions under which they are applicable.

We present a system that is able to learn continuous operators through experience and use them to improve the performance of its task. Our system performs autonomous robot navigation in unstructured and unknown terrains. It consists of a reactive navigation module which is continuously tuned by an adaptation and learning module according to sensory information perceived by the system. As the system navigates through a terrain, the adaptation and learning module compiles its experiences into reliable mappings between input sensory information and control outputs, and uses these mappings to modify the behavior of the reactive module. These mappings represent the "continuous operators" discussed above.

The reader is referred to Ram and Santamaria (1993a) for a detailed explanation of the system and for the description of the evaluation procedures used to verify the validity of our approach. In this paper, we provide a brief description of the problem domain and the methods used for acting, adapting, and learning, focussing on the speedup learning and knowledge compilation aspects of the research. It is interesting to note that our system is not only able to perform speedup learning, that is, to reduce the time taken by the performance task, but it can also improve the "quality" of the performance task as well. Since problem solving occurs continuously during the performance of the task, improving the quality of the performance usually results in speeding up the problem solving time as well. In robot navigation, for example, reducing the number of collisions during navigation (a qualitative improvement) results in a reduction in the total time spent on reactive control of the robot.

This paper is organized as follows. Section 2 provides a general description of the architecture of the system. Section 3 describes the operation of the learning and adaptation module, which is responsible for the speedup learning and knowledge compilation of continuous operators. Section 4 describes our evaluation of the system, and section 5 concludes the paper.

2 SYSTEM DESCRIPTION

The Self-Improving Navigation System (SINS) consists of a navigation module, which uses schema-based reactive control methods, and an on-line adaptation and learning module, which uses case-based reasoning and reinforcement learning methods. The navigation module is responsible for moving the robot through the terrain from the starting location to the desired goal location while avoiding obstacles along the way. A set of control parameters can be used to change the behavior of the navigation module. The adaptation and learning module is responsible for changing the behavior of the navigation module in such a way that the performance of the navigation task is improved. In particular, the adaptation and learning module constructs mappings from sensory input information to appropriate control parameters, which are then used to modify the behavior of the navigation module in an effective manner. These mappings are represented as "cases" that encapsulate the system's navigational experiences, and correspond to what one might think of as the "continuous operators" for this problem domain.

The learning and adaptation module, therefore, performs two main functions. The adaptation sub-module performs on-line adaptation of the navigation module by continuously supplying the appropriate control parameters required to get the best performance from the navigation module. The adaptation is based on recommendations from cases that contains specific mappings between sensory input and control outputs. With such mappings, SINS is able to predict future consequences of its actions and act appropriately. The learning sub-module monitors the progress of the system and incrementally modifies the content of the cases through experience so that they are able to provide more reliable recommendations on how to adapt the navigation module in the future. Figure 1 shows the SINS functional architecture.

The navigation module is based on the AuRA architecture (Arkin, 1989), and consists of a set of motor schemas that represent the individual motor control behaviors available to the system. Each schema reacts to sensory information from the environment, and produces a velocity vector representing the direction and speed at which the robot is to move given current environmental conditions. The velocity vectors produced by all the schemas are then combined to produce a velocity vector (or potential field) that directs

[1] While we have used problem-space search terminology in this discussion to emphasize the similarities with the standard techniques for speedup learning and knowledge compilation, it should be noted that continuous states and operators are different in several fundamental respects from the more familiar discrete states and operators, in particular, in the details of the methods for learning, adapting, matching, and acting.

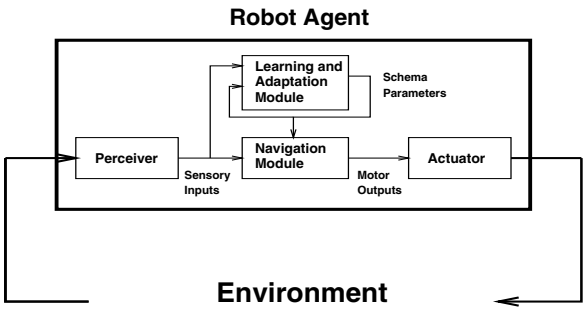

Figure 1: System architecture.

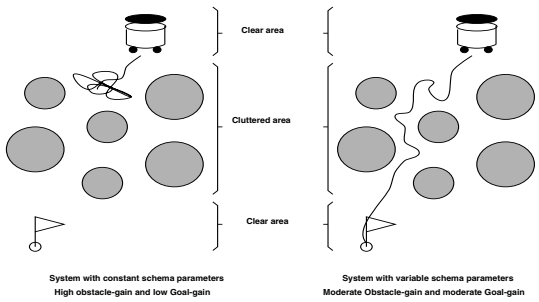

Figure 2: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing it to "squeeze" through the obstacles and then take a relatively direct path to the goal.

the actual movement of the robot. Simple behaviors, such as wandering, obstacle avoidance, and goal following, can combine to produce complex emergent behaviors in a particular environment. Different emergent behaviors can be obtained by combining and modifying the simple behaviors.

A detailed description of schema-based reactive control methods can be found in Arkin (1989). For the research presented in this paper, we used three motor schemas: AVOID-STATIC-OBSTACLE, MOVE-TO-GOAL, and NOISE. AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles. MOVE-TO-GOAL schema directs the system to move towards a particular point in the terrain. The NOISE schema makes the system wander in a random direction. Each motor schema has a set of parameters that controls the velocity vector generated by the motor schema. For this paper, we used the following parameters: **Obstacle-Gain**, associated with AVOID-STATIC-OBSTACLE, determines the magnitude of the repulsive field generated by the obstacles perceived by the system; **Goal-Gain**, associated with MOVE-TO-GOAL, determines the magnitude of the attractive field generated by the goal; **Noise-Gain**, associated with NOISE, determines the magnitude of the wandering behavior; and **Noise-Persistence**, also associated with NOISE, determines the duration for which a noise value is allowed to persist.

Different combinations of schema parameters produce different navigational behaviors (see figure 2). Traditionally, schema parameters have been fixed and determined ahead of time by the system designer. However, on-line selection and modification of the appropriate parameters based on the current environmental situation can enhance navigational performance, as in the ACBARR system (Ram, Arkin, Moorman, & Clark, 1992). ACBARR can dynamically modify its behavioral parameters in order to perform successfully in different environmental configurations requiring different navigational "strategies". SINS adopts this approach by allowing schema parameters to be modified continuously and dynamically. However, in ACBARR, navigational strategies are supplied ahead of time by the designer using hand-coded "cases" which specify the appropriate parameter values for different environmental situations. SINS, in contrast, can learn and modify its own cases through experience. The representation of cases in SINS is also considerably different and is designed to support continuous adaptation and learning; in particular, the representation encodes continuous operators that are constructed and modified through experience.

3 LEARNING AND ADAPTATION MODULE

The navigation module in SINS can be adapted to exhibit many different behaviors. The learning and adaptation module improves the performance of the system by learning how and when to tune the navigation module. In this way, the system can use the appropriate behaviors in each environmental situation encountered during navigation, and can continuously tune these behaviors as it moves through the terrain. The learning module, therefore, must learn about and discriminate between different environments, and associate with each the appropriate adaptations to be performed on the motor schema parameters. This requires a representational scheme to model, not just the environment (as in, say, a map-learning system), but the interactions between the system and the environment. In addition, to ensure that the system does not get bogged down in extensive high-level reasoning, the knowledge represented in the model must be based on perceptual and motor information easily available at the reactive level.

The learning and adaptation module uses a combination of ideas from case-based reasoning and learning, which deals with the issue of using past experiences to deal with and learn from novel situations (e.g., see Kolodner, 1993; Hammond, 1989), and from reinforcement learning, which deals with the issue of updating the content of system's knowledge based on feedback from the environment (e.g., see Sutton, 1992). SINS starts out with no knowledge about how and when to modify the schema parameter values, relying purely on reactive control to navigate through the terrain. While navigating, the system identifies useful associations between sensory inputs and schema parameter values by applying sequences of schema parameter values that have worked in the past (initially, schema parameter values are adapted randomly), and remembering the results that these sequences produce. Sequences that produce consistent input-output associations are reinforced. The method is similar to case-based reasoning in the sense that

new sequences of associations are learned by remembering and adapting previous sequences that have been used in the past in similar environmental situations. The method is also similar to reinforcement learning in the sense that it tends to reinforce and perform those actions that are considered "good" according to a utility reward metric.

It is the combination of case-based reasoning and reinforcement learning that allows the system to remember its experiences and to identify and reinforce the useful aspects of these experiences. However, the combination, and the nature of the continuous task domain, required us to deviate from the standard algorithms for case-based reasoning and reinforcement learning (see Ram & Santamaria, 1993b, for a discussion of our method for "continuous case-based reasoning"). The main responsibility of the case-based reasoning method is to (continuously) tune the navigation module using control outputs based on the (continuous) sequences of control outputs used in past similar situations. The main responsibility of the reinforcement learning method is to (continuously) adapt these sequences based on the results of applying them to the current situation, so that they provide better predictions in future situations. The system is able to improve its navigational performance because the case-based method directs the system to use similar sequences of schema parameter values under similar environmental situations, while the reinforcement learning directs the system to remember only those mappings that were successful according to a utility reward metric. With experience, the system is able to capture and apply only those sequences of associations that actually improve the performance of the navigation task; these associations, then, come to represent the continuous operators for this task domain and can be used across a wide range of navigational terrains.

The learning and adaptation module uses a model consisting of sequence of associations between the sensory inputs and schema parameters values. Each sequence of associations is represented as a case. Sensory inputs provide information about the environmental situation, and schema parameter information specifies how to adapt the values of the control parameters used by the navigation module in the environments to which the case is applicable. Each type of information is represented as a vector of analog values. Each analog value corresponds to a quantitative variable (a sensory input or a schema parameter) at a specific time. A vector represents the trend or recent history of a variable. A case captures a sequence of associations between sensory inputs and schema parameters by grouping their respective vectors together. Thus a case encodes what we earlier called a continuous macro-operator, since each single association can be considered as a continuous operator that, if applied under its specified environmental situation, should produce the environmental situation encoded in the next association of the sequence. Figure 3 show an example of this representation. Note that the applicability conditions of these continuous operators do not specify just the current environmental "state" but rather the recent history or trend of the individual state descriptors.

For the results presented in this paper, we used four input vectors to characterize the environment and discriminate among different environmental situations: **Obstacle-Density** provides a measure of the occupied areas that impede navigation; **Absolute-Motion** measures the activity of the system; **Relative-Motion** represents the change in motion activity; and **Motion-Towards-Goal** specifies how much progress the system has actually made towards the goal. These input vectors are constantly updated with the information received from the sensors.

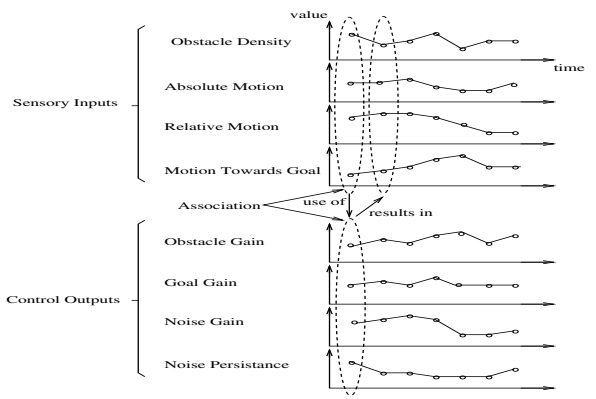

Figure 3: Sample representations showing the time history of analog values representing perceived inputs and schema parameters. Associations between sensory inputs and control outputs are arranged vertically and the sequence of associations are arranged horizontally.

We also used four output vectors to represent the schema parameter values used to adapt the navigation module, one for each of the schema parameters (**Obstacle-Gain**, **Goal-Gain**, **Noise-Gain**, and **Noise-Persistence**) discussed earlier. The values are set periodically according to the recommendations of the case that best matches the current sequence of environmental situations. The new values remain constant over a "control interval" until the next setting period.

While solving a navigation problem, SINS keeps track of a history of the sequence of environmental situations and schema parameter values used in the recent past. After each control interval, the case-based reasoning method retrieves the case encoding the sequence of associations most similar to the current one. At that point, the learning and adaptation module tends to select the schema parameter values that the next association in the sequence suggests, which are handed to the reactive navigation module. The selection of schema parameter values is performed using a probabilistic function based on a utility reward and on the schema parameter values suggested by the next association in the case. The function is similar to the corresponding function used in standard reinforcement learning methods, that is, the probability that the selection of schema parameter values is similar to the ones encoded in the next association in the case increases with the reward utility. After the schema parameters values have been applied, the learning and adaptation module watches the results (i.e., the resulting environmental situation) and adapts the association in the case that was just used based on the extent to which its predictions were met. Technical details of the learning

methods can be found in Ram and Santamaria (1993a).

One difference in our methods from standard reinforcement learning is that SINS uses a utility reward that only used "internal" information about the operation of the system. That is, we do not encode explicit information in the utility reward which signals that goals are "good" and obstacles are "bad". Instead, the utility reward is based on a "relative similarity metric" which measure quantifies how similar the current sequence of associations is to the sequence of associations encoded by the case relative to how similar the case's sequence of associations has been in previous utilizations of the case. Intuitively, if a case matches the current situation better than it matched previous situations it was used in, it is likely that the sequence of associations encoded in the case will provide good predictions in situations similar to the current one; thus, it is worthwhile reinforcing the associations in the case in the direction of the current association because the current association is representative of the class of situations to which the case should be applied. Alternatively, if the match is not quite as good, the case should not be modified because the predictions will not be quite as good in future similar situations; although it may still be used to guide performance if no better case can be found, it is likely that the continuous operators that the case is beginning to capture are not the ones that ought to be refined based on this experience.

Since the reinforcement formula is based on a relative similarity measure, the overall effect of the learning process is to cause the cases to converge on stable associations between environment configurations and schema parameters. Stable associations represent regularities in the world that have been identified by the system through its experience, and provide the predictive power necessary to navigate in future situations. The assumption behind this method is that the interaction between the system and the environment can be characterized by a finite set of causal patterns or associations between the sensory inputs and the behavioral parameters that define the actions performed by the system. The method allows the system to learn these causal patterns and to use them to modify its actions by updating its schema parameters as appropriate. The system improves its performance, not because it is being reinforced directly for reaching the goal, but because it has a tendency to replicate and reinforce consistent behavior. Since the motor schemas used in the navigational module have an implicit goal to move the robot towards the goal while avoiding obstacles, the system gets better at performing that task because its cases capture and use the sequence of associations that involve just those causal patterns that are used for the task. In other words, SINS is simply learning all the input-output patterns that correspond to the continuous operators in its domain; improvement in performance results from the fact that the navigation module generates what might be thought of as good "training data" for the learning module.

4 EXPERIMENTS AND RESULTS

The methods presented above have been evaluated using extensive simulations across a variety of different types of environment, performance criteria, and system configurations. The objective of these experiments is to measure qualitatively and quantitatively improvement of the navigation performance of SINS, and to compare this performance against a non-learning schema-based reactive system (the "static system") and a system that changes the schema parameter values randomly after every control interval (the "random system"). We chose these systems because they represent successive stages in the complexity of a complete navigational system. The static system cannot adapt the schema parameter values as it navigates, which makes it efficient only on those environmental situation for which the schema parameters were explicitly designed. If the environment changes too widely, the system's performance deteriorates rapidly. The random system is able to cope with environmental changes because it can randomly happen to chance upon the appropriate schema parameter values, but it is not able to compile knowledge from experiences it has had in the past that could be useful in the current situation. In effect, every time the random system faces a new situation it must search, using random selection, for a specific combination of schema parameter values that can be useful for the current situation. Finally, SINS can not only change the schema parameter values, it can also compile knowledge in terms of sequences of associations that have been successful in the past and use these sequences directly to improve its performance on the task. In this way, the system avoids the search for specific combinations of schema parameter values in new situations, resulting directly in performance speedup.

The three systems were tested on randomly generated environments consisting of rectangular bounded worlds. Each environment contains circular obstacles, a start location, and a destination location, as shown in figure 2. The location, number, and radius of the obstacles were randomly determined to create environments of varying amounts of *clutter*, defined as the ratio of free space to occupied space. We tested the effect of three design parameters in SINS: *max-cases*, the maximum number of cases that SINS is allowed to create; *case-length*, representing the maximum number of associations in a case; and *control-interval*, which determines how often the schema parameters in the reactive control module are adapted.

We used six estimators to evaluate the navigation performance of the systems. These metrics were computed using a cumulative average over the test worlds to factor out the intrinsic differences in difficulty of different worlds. *Average number of worlds solved* indicates in how many of the worlds posed the system actually found a path to the goal location. The optimum value is 100% since this would indicate that every world presented was successfully solved. *Average steps* indicates the average of number of steps that the robot takes to terminate each world; smaller values indicate better performance. *Average distance* indicates the total distance traveled per world on average; again, smaller values indicate better performance. *Average* $\frac{actual}{optimal}$ *distance per world* indicates the ratio of the total distance traveled and the Euclidean distance between the start and end points, averaged over the solved worlds. The optimal value is 1, but this is only possible in a world without ob-

Figure 4: Final performance results

	static	*random*	*adaptive (SINS)*
Worlds solved	14.4%	41.3%	94.5%
Steps per world	2626.5	2051.5	708.8
Distance per world	350.2	697.9	216.9
$\frac{\text{Actual}}{\text{Optimal}}$ distance	8.6	17.2	5.5
Virtual collisions	45.9	26.3	26.3
Time per world (ms)	2946.7	2359.4	1757.2

stacles. *Average virtual collisions per world* indicates the total number of times the robot came within a pre-defined distance of an obstacle. Finally, *average time per world* indicates the total time the system takes to execute a world on average.

The data for the estimators was obtained after the systems terminated each world. This was to ensure that we were consistently measuring the effect of learning across experiences rather than within a single experience (which is less significant on worlds of this size anyway). The execution is terminated when the navigation system reaches its destination or when the number of steps reaches an upper limit (3000 in the evaluation presented here). The latter condition guarantees termination since some worlds are unsolvable by one or more systems.

Detailed experimental results may be found in Ram and Santamaria (1993a). Table 4 shows the final results after the execution of 200 randomly generated worlds, based on the design parameters *max-cases* $= 10$, *max-cases* $= 4$, and *control-interval* $= 12$. SINS successfully navigates almost 95% of the worlds, a 555% improvement over the static system, with 43% fewer virtual collisions, 271% fewer steps, 38% shorter distance travelled, 36% better $\frac{\text{actual}}{\text{optimal}}$ ratio, and 40% speedup in performance time. SINS also solves each world on average 20% faster than the random system.

Figure 5 shows the history of the performance metrics over successive problem worlds to demonstrate the speedup performance of SINS. Note that SINS improves its performance even though it is working on different problems each time (recall that each of the 200 worlds were randomly generated). This is because the system is not learning about a particular world but rather is learning continuous operators for navigation that are generally applicable to a wide range of possible navigational problems. Since not all the worlds have the same level of "difficulty", the performance figures are not strictly monotonic and show peaks at different points; however, the tendency towards cumulative performance improvement is evident in the graphs.

5 Conclusions

SINS combines and extends case-based reasoning and reinforcement learning methods to provide a basis for speedup learning in continuous domains. Learning corresponds to building a library of experiential knowledge that can help the system improve its performance. Since the system is always learning, it can cope with major environmental changes as well as fine tune its navigation module in static and specific environment situations. The learning methods in SINS represent a novel kind of knowledge compilation and speedup learning for continuous domains. Knowledge compilation occurs through the compilation of continuous experiences into higher-level (but still continuous) knowledge structures (representing continuous operators and macro-operators) that guide performance in future situations. This may be thought of as a kind of constructive representational change in which raw sensorimotor experiences are compiled into higher-level operators for navigation (Ram, 1993). The learning methods result in improvements in the quality of performance in addition to its speed, which are realized even when the system is faced with a different problem situation each time.

References

Arkin, R.C. (1989). Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92-112.

DeJong, G.F. & Mooney, R.J. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(1):145-176.

Fikes, R.E., Hart, P.E., & Nilsson, N.J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251-288.

Hammond, K.J. (1989). *Case-Based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.

Kolodner, J.L. (1993). *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA (in press).

Laird, J.E., Rosenbloom, P.S., Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11-46.

Minton, S.N. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363-392.

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(2):47-80.

Mitchell, T. M., Utgoff, P. E. & Banerji, R. B. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. I), Morgan Kaufmann, San Mateo, CA.

Ram, A. Creative Conceptual Change. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO, 1993.

Ram, A., Arkin, R.C., Moorman, K. & Clark, R.J. (1992). Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotics systems. Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology, Atlanta, GA.

Ram, A. & Santamaria, J.C. (1993a). A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robotic navigation. In *Proceedings of the Second International Workshop on Multistrategy Learning*, Harper's Ferry, WV.

Ram, A. & Santamaria, J.C. (1993b). Continuous case-based reasoning. In *Proceedings of the AAAI Workshop on Case-Based Reasoning*, Washington, DC (to appear).

Sutton, R.S. (1992), editor. *Machine Learning*, 8(3/4), special issue on Reinforcement Learning, 1992.

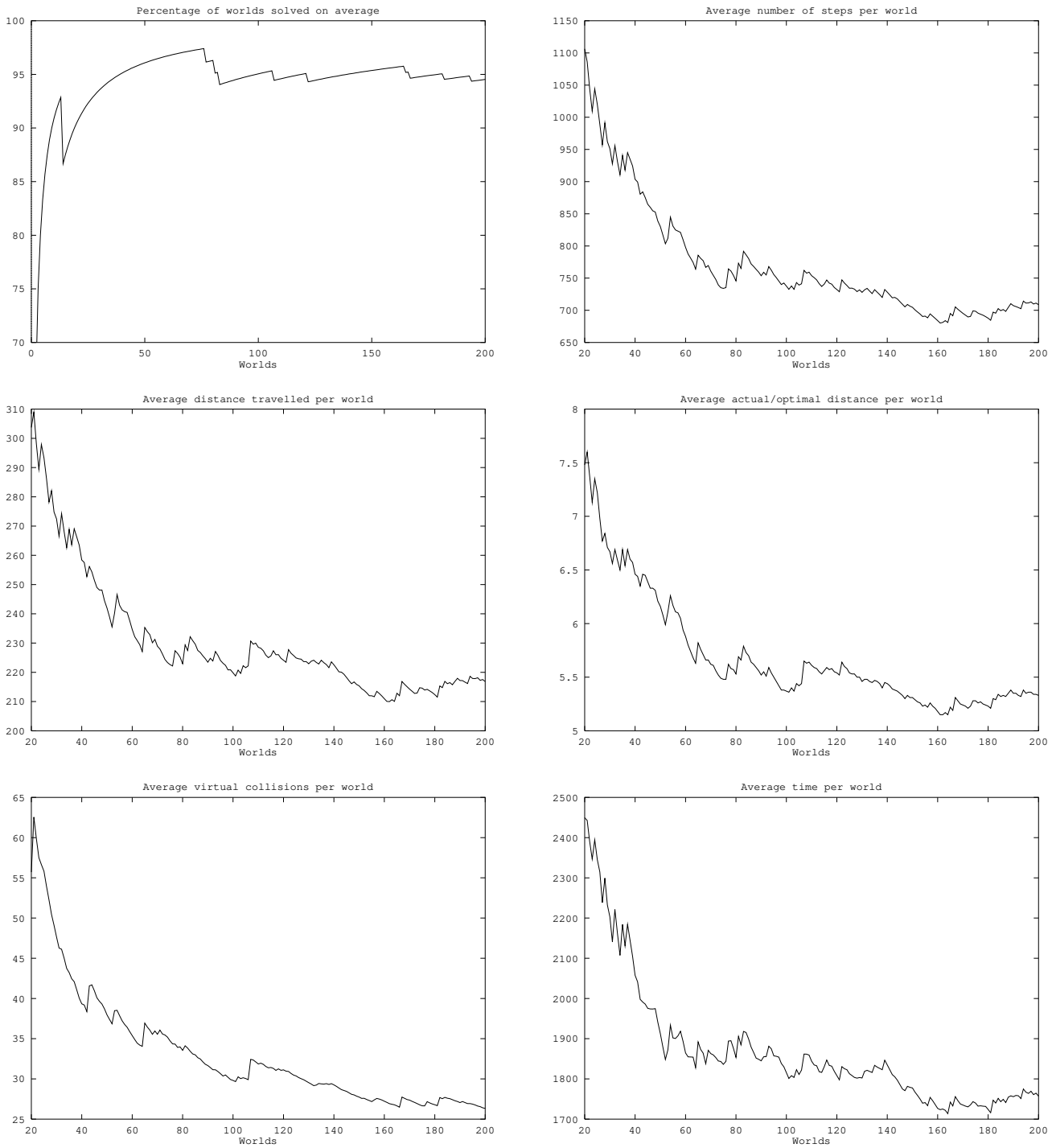

Figure 5: Graphs showing improvement in performance with experience.