Proceedings of the AAAI-93 Workshop on Case-Based Reasoning, pp. 86–93, Washington, DC, July 1993.

Continuous Case-Based Reasoning

Ashwin Ram and Juan Carlos Santamaría
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
{ashwin,carlos}@cc.gatech.edu

Abstract

Case-based reasoning systems have traditionally been used to perform high-level reasoning in problem domains that can be adequately described using discrete, symbolic representations. However, many real-world problem domains, such as autonomous robotic navigation, are better characterized using continuous representations. Such problem domains also require continuous performance, such as continuous sensorimotor interaction with the environment, and continuous adaptation and learning during the performance task. We introduce a new method for *continuous case-based reasoning*, and discuss how it can be applied to the dynamic selection, modification, and acquisition of robot behaviors in autonomous navigation systems. We conclude with a general discussion of case-based reasoning issues addressed by this work.

1 Introduction

Case-based reasoning systems have traditionally been used to perform high-level reasoning in problem domains that can be adequately described using discrete, symbolic representations. For example, CHEF uses case-based planning to create recipes [Hammond, 1989], AQUA uses case-based explanation to understand newspaper stories [Ram, 1993b], HYPO uses case-based interpretation for legal argumentation [Ashley & Rissland, 1987], MEDIATOR uses case-based problem solving for dispute resolution [Kolodner, Simpson, & Sycara, 1985], and PRODIGY uses case-based reasoning in the form of derivational analogy for high-level robot planning [Veloso & Carbonell, 1991].

In our research, we have been investigating the problem of performance and learning in continuous, real-time problem domains, such as autonomous robotic navigation. Our method, called *continuous case-based reasoning* shares many of the fundamental assumptions of what might be called "discrete" case-based reasoning in symbolic problem domains.[1] Learning is integrated with performance. Performance is guided by previous experience. New problems are solved by retrieving cases and adapting them. New cases are learned by evaluating proposed solutions and testing them on a real or simulated world. However, continuous problem domains require different underlying representations and place additional constraints on the problem solving process. For example, consider the problem of driving a car on a highway. Car driving experiences can vary from one another in infinitely many ways. The speed of a car might be 55 mph in one experience and 54 mph in another. Within a given episode, the speed of the car might continuously vary, both infinitesimally from moment to moment, and significantly from, say, the highway to an exit ramp. The problem solving and learning process must operate continuously; there is no time to stop and think, nor a logical point in the process at which to do so.

Such problem domains are "continuous" in three senses. First, they require *continuous representations*. For example, a robotic navigation task requires representations of continuous perceptual and motor control information. Second, they require *continuous performance*. For example, driving a car requires continuous action. Often, problem-solving performance is incremental of necessity because of limited knowledge available to the reasoning system and/or because of the unpredictability of the environment; the system can at best execute the "best" short-term actions available to it and then re-evaluate its progress. A robot, for example, may not know where obstacles lie until it actually encounters them. Third, these problem domains require *continuous adaptation and learning*. As the problems encountered become more varied and difficult, it becomes necessary to use fine-grained, detailed knowledge in an incremental manner to act, and to rely on continuous feedback from the environment to adapt actions and learn from experiences.

Case-based reasoning in such problem domains requires significant enhancements to the basic case-based reasoning methods used in discrete, symbolic reasoning systems. When are two experiences different enough to warrant consideration as independent cases? What is the scope of a single case? Is the entire car trip from one's house to the grocery store a single case that can be used to guide and improve one's driving performance in future situations? How should "continuous cases" be represented? How can they be used to guide performance? How are they learned and modified through experience? And how can this performance and learning be integrated into a continuous, on-line, real-time process?

In this paper, we provide an answer to these questions based on our research into a robot navigation task. The proposed methods are fully implemented in a computer system which uses reactive control for its performance element, and case-based reasoning for continuous adaptation of the performance element and for continuous learning through experience. We discuss the relevant technical details of the system, and we conclude with the contributions of our research and their implication for the design of case-based reasoning systems in general.

[1]We do not eschew symbolic representations; rather, the issue is the continuous, time-varying nature of any proposed representations, whether symbolic, numeric, or otherwise.

2 The robot navigation task

Autonomous robotic navigation is defined as the task of finding a path along which a robot can move safely from a source point to a destination point in a obstacle-ridden terrain, and executing the actions to carry out the movement in a real or simulated world. Several methods have been proposed for this task, ranging from high-level planning methods to reactive methods. High-level planning methods use extensive world knowledge about available actions and their consequences to formulate a detailed plan before the actions are actually executed in the world (e.g., [Fikes, Hart, & Nilsson, 1972; Georgeff, 1987; Maes, 1990; Sacerdoti, 1977]). Considerable high-level knowledge is also needed to learn from planning experiences (e.g., [Hammond, 1989; Minton, 1988; Mostow & Bhatnagar, 1987; Segre, 1988]). Situated or reactive control methods, in contrast, perform no planning; instead, a simple sensory representation of the environment is used to select the next action that should be performed [Arkin, 1989; Brooks, 1986; Kaelbling, 1986; Payton, 1986]. Actions are represented as simple behaviors, which can be selected and executed rapidly, often in real-time. These methods can cope with unknown and dynamic environmental configurations, but only those that lie within the scope of predetermined behaviors. Furthermore, such methods cannot modify or improve their behaviors through experience, since they do not have any predictive capability that could account for future consequences of their actions, nor a higher-level formalism in which to represent and reason about the knowledge necessary for such analysis.

We have developed a self-improving navigation system that uses reactive control for fast performance, augmented with a continuous case-based reasoning method that allow the system to adapt to novel environments and to learn from its experiences. The system autonomously and progressively constructs representational structures that aid the navigation task by supplying the predictive capability that standard reactive systems lack. The representations are constructed using a hybrid case-based and reinforcement learning method without extensive high-level reasoning. The system is very robust and can perform successfully in (and learn from) novel environments, yet it compares favorably with traditional reactive methods in terms of speed and performance. A further advantage of the method is that the system designers do not need to foresee and represent all the possibilities that might occur since the system develops its own “understanding” of the world and its actions. Through experience, the system is able to adapt to, and perform well in, a wide range of environments without any user intervention or supervisory input. This is a primary characteristic that autonomous agents must have to interact with real-world environments.

3 Technical details

Our self-improving robotic navigation system 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 environment from the starting location to the desired goal location while avoiding obstacles along the way. The adaptation and learning module has two responsibilities. The adaptation sub-module performs on-line adaptation of the reactive control parameters to get the best performance from the navigation module. The adaptation is based on recommendations from cases that capture and model the interaction of the system with its environment. With such a model, the system is able to predict future consequences of its actions and act accordingly. The learning sub-module monitors the progress of the system and incrementally modifies the case representations through experience. Figure 1 shows the functional architecture of the system.

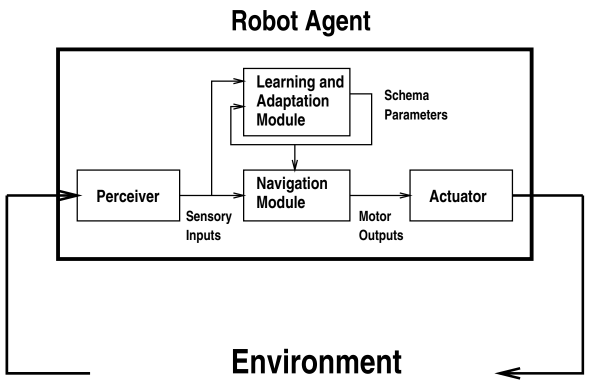

Figure 1: System architecture

3.1 Schema-based reactive control

The reactive control module is based on the AuRA architecture [Arkin, 1989], and consists of a set of motor schemas that represent the individual motor 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. For example, the schema AVOID-STATIC-OBSTACLE directs the system to move itself away from detected obstacles, and the associated schema parameter **Obstacle-Gain** determines the magnitude of the repulsive potential field generated by the obstacles perceived by the system. The velocity vectors produced by all the schemas are then combined to produce a potential field that directs 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 modifying the simple behaviors. A detailed description of schema-based reactive control methods can be found in Arkin [1989].

3.2 Behavior selection and modification

Our first attempt at building a case-based reactive navigation system focussed on the issue of using case-based reasoning to guide reactive control. A central issue here is the nature of the guidance: At what grain size should the reactive control module represent its behaviors, and what kind of “advice” should the case-based reasoning module provide to the reactive control module?

In order to achieve more robust robotic control, we advocate the use of sets of behaviors, called *behavior assemblages*, to represent appropriate collections of cooperating behaviors for complex environments, and of *behavior adaptation* to adapt and fine-tune existing behaviors dynamically in novel environments [Ram, Arkin, Moorman, & Clark, 1992]. There are two types of behavior adaptations that might be considered. One option is to have the system modify its current behavior based on immediate past experience. While useful, this is only a local response to the problem. A more global solution is to have the system select completely new assemblages of behaviors based on the current environment in which it finds itself. A robust system should be able to learn about and adapt to its environment dynamically in both these ways.

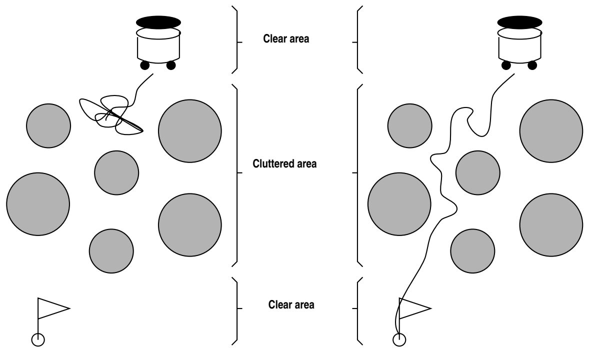

Figure 2: Typical navigational behaviors of different tunings of the reactive control module. The figure on the left shows the non-learning system with high obstacle avoidance and low goal attraction. On the right, the learning system has lowered obstacle avoidance and increased goal attraction, allowing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

Different combinations of schema parameters produce different behaviors to be exhibited by the system (see figure 2). This allows the system to interact successfully in different environmental configurations requiring different navigational "strategies" [Clark, Arkin, & Ram, 1992; Moorman & Ram, 1992]. Traditionally, parameters are fixed and determined ahead of time by the system designer. However, on-line selection and modification of the appropriate parameters based on the current environment can enhance navigational performance. We tested this idea by building the ACBARR (A Case-BAsed Reactive Robotic) system and evaluating it qualitatively and quantitatively through extensive simulation studies using a variety of different environments and several different performance metrics (see [Ram, Arkin, Moorman, & Clark, 1992] for details). The experiments show that ACBARR is very robust, performing well in novel environments. Additionally, it is able to navigate through several "hard" environments, such as box canyons, in which traditional reactive systems would perform poorly.

In the ACBARR system, we incorporated both *behavior adaptation* and *behavior switching* into a reactive control framework. At the local level, this is accomplished by allowing the system to adapt its current behavior in order to build "momentum". If something is working well, the system continues doing it and tries doing it a little bit harder; conversely, if things are not proceeding well, the system attempts something a little different. This technique allows the system to fine tune its current behavior patterns to the exact environment in which it finds itself. For example, if the robot has been in an open area for a period of time and has not encountered any obstacles, it picks up speed and does not worry as much about obstacles. If, on the other hand, it is in a cluttered area, it lowers its speed and treats obstacles more seriously. For behavior-based reactive systems, this translates into altering the schema gains and parameters continuously, provided the system has a method for determining the appropriate modifications. ACBARR uses case-based reasoning to retrieve behavior modification rules. These rules are then used incrementally to alter the gain and parameter values based on current environmental conditions and past successes.

The other method for behavior modification in ACBARR is at a more global level. If the system is currently acting under the control of an assemblage of behaviors which are no longer suited to the current environment, it selects a new assemblage based on what the environment is now like. Continuing with the above example, suppose that the robot is in a very cluttered environment and is employing a conservative assemblage of motor behaviors. It then breaks out of the obstacles and enters a large open field (analogous to moving from a forested area into a meadow). If only local changes were allowed, the robot would eventually adjust to the new environment. However, by allowing a global change to take place, the system needs only to realize that it is in a radically new environment and to select a new assemblage of motor behaviors, one better suited to the new surroundings. Interestingly, case-based reasoning is used to realize this type of modification as well.

Assemblages of behaviors are represented as *cases*, or standard scenarios known to the system, that can be used to guide performance in novel situations. As in a traditional case-based reasoning system, a case is used to propose a plan or a solution (here, a behavior assemblage) to the problem (here, the current environmental configuration). However, our method differs from the traditional use of case-based reasoning in an important respect. A case in our system is also used to propose a set of *behavior adaptations*, rather than merely the behaviors themselves. This allows the system to use different strategies in different situations. For example, the system might use a "cautious" strategy in a crowded environment by gradually slowing down and allowing itself to get closer to the surrounding obstacles. In order to permit this, strategies suggest boundaries on behavioral parameters rather than precise values for these parameters. Cases are used both to suggest behavior assemblages as well as to perform dynamic (on-line) adaptation of the parameters of behavior assemblages within the suggested boundaries. The knowledge required for both kinds of suggestions is stored in a case, in contrast with traditional case-based reasoning systems in which cases are used only to suggest solutions, and a separate library of adaptation rules is used to adapt a solution to fit the current problem. Further details can be found in Ram, Arkin, Moorman & Clark [1992].

3.3 Case representation

While ACBARR demonstrated the feasibility of on-line, case-based reasoning in systems requiring continuous, real-time response, it relied on a fixed library of cases that were hand-coded into the system. The system could adapt to novel environments—an important kind of learning—but it could not improve its own adaptation behavior through experience. Since the knowledge required for behavior adaptation is stored in cases, we turned our attention to the problem of learning cases through experience. We built SINS (Self-Improving Navigation System), which is similar to the ACBARR system but can learn and modify its own cases through experience. The representation of the cases in SINS is different and is designed to support learning, but the underlying ideas behind the two systems are very similar.

Since we did not want to rely on hand-coded, high-level domain knowledge, the representations used by SINS to model its interaction with the environment are initially under-constrained and generic; they contain very little useful information for the navigation task. As the system interacts with the environment, the learning module gradually modifies the content of the representations until they become useful and provide reliable information for adapting the navigation system to the particular environment at hand.

The learning and reactive modules function in an integrated manner. The learning module is always trying to find a better model of the interaction of the system with its environment so that it can tune the reactive module to perform its function better. The reactive module provides feedback to the learning module so it can build a better model of this interaction. The behavior of the system is then the result of an equilibrium point established by the learning module which is trying to refine the model and the environment which is complex and dynamic in nature. This equilibrium may shift and need to be re-established if the environment changes drastically; however, the model is generic enough at any point to be able to deal with a very wide range of environments.

The reactive module in SINS can be adapted to exhibit many different behaviors. SINS improves its performance by learning how and when to tune the reactive module. In this way, the system can use the appropriate behavior in each environmental configuration encountered. 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 schemas. This requires a representational scheme to model the interaction between the system and the environment. However, 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.

SINS uses a model consisting of associations between the sensory inputs (e.g., **Obstacle-Density**) and schema parameters values (e.g., **Obstacle-Gain**, associated with the AVOID-STATIC-OBSTACLE schema). Each set of associations is represented as a case. Sensory inputs provides information about the configuration of the environment, and is obtained from the system's sensors. Schema parameter information specifies how to adapt the reactive 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 models an association between sensory inputs and schema parameters by grouping their respective vectors together. Figure 3 show an example of this representation.

This representation has three essential properties. First, the representation is capable of capturing a wide range of possible associations between of sensory inputs and schema parameters. Second, it permits continuous progressive refinement of the associations. Finally, the representation captures trends or patterns of input and output values over time. This allows the system to detect patterns over larger time windows rather than having to make a decision based only on instantaneous values of perceptual inputs.

3.4 Case learning

The case-based reasoning and learning module creates, maintains and applies the case representations used for on-line adaptation of the reactive module. The main objective of the learning method is to construct a model of the continuous sensorimotor interaction of the system with its environment, that is, a mapping from sensory inputs to appropriate behavioral (schema) parameters. This model allows the adaptation module to continuously control the behavior of the navigation module by selecting and adapting schema parameters in different environments. To learn a mapping in this context is to detect and discriminate among different environment configurations, and to identify the appropriate schema parameter values to be used by the reactive module, in a dynamic and on-line manner. This means that, as the system is navigating, the learning module is perceiving the environment, detecting an environment configuration, and modifying the schema parameters of the reactive module

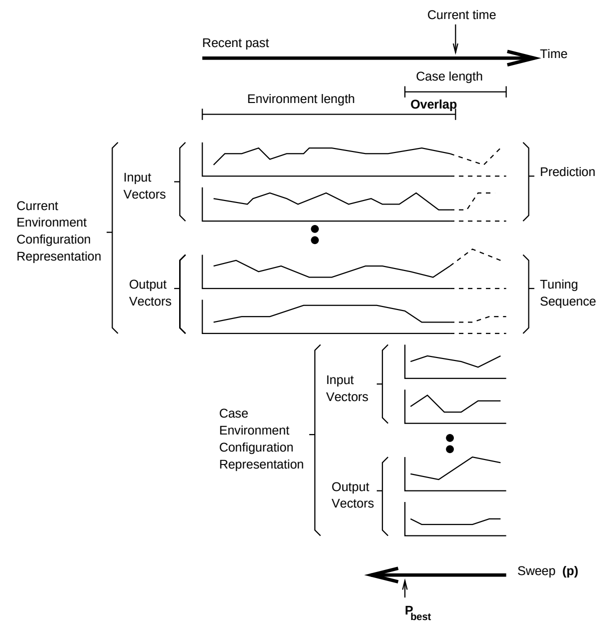

Figure 3: Sample representations showing the time history of analog values representing perceived inputs and schema parameters. Each graph in the case (below) is matched against the corresponding graph in the current environment (above) to determine the best match, after which the remaining part of the case is used to guide navigation (shown as dashed lines).

accordingly, while simultaneously updating its own cases to reflect the observed results of the system's actions in various situations.

The method is based on 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, and from reinforcement learning, which deals with the issue of updating the content of system's knowledge based on feedback from the environment (see [Sutton, 1992]). However, in traditional case-based planning systems (e.g., [Hammond, 1989]) learning and adaptation requires a detailed model of the domain. This is exactly what reactive planning systems are trying to avoid. Earlier attempts to combine reactive control with classical planning systems (e.g., [Chien, Gervasio, & DeJong, 1991]) or explanation-based learning systems (e.g., [Mitchell, 1990]) also relied on deep reasoning and were typically too slow for the fast, reflexive behavior required in reactive control systems. Unlike these approaches, our method does not fall back on slow non-reactive techniques for improving reactive control.

Each case represents an observed regularity between a particular environmental configuration and the effects of different actions, and prescribes the values of the schema parameters that are most appropriate (as far as the system knows based on its previous experience) for that environment. The learning module performs the following tasks in a cyclic manner: (1) **perceive** and represent the current environment; (2) **retrieve** a case whose sensory input vector represents an environment most similar to the current environment; (3) **adapt** the schema parameter values in use by the reactive control module by installing the values recommended by schema parameter vectors of the case; and (4) **learn** new associations and/or adapt existing associations represented in the case to reflect any new information gained through the use of the case in the new situation to enhance the reliability of their predictions.

The **perceive** step builds a set of vectors representing the sensory input, which are then matched against the corresponding vectors of the cases in the system's memory in the **retrieve** step. The case similarity metric is based on the mean squared difference between each of the vector values of the case over a trending window, and the vector values of the environment. The best match window is calculated using a reverse sweep over the time axis similar to a convolution process to find the relative position that matches best. The best matching case is handed to the **adapt** step, which selects the schema parameter values from the case and modifies the corresponding values of the reactive behaviors currently in use using a reinforcement formula which uses the case similarity metric as a scalar reward. Thus the actual adaptations performed depend on the goodness of match between the case and the environment.

Finally, the **learn** step uses statistical information about prior applications of the case to determine whether information from the current application of the case should be used to modify this case, or whether a new case should be created. The vectors encoded in the cases are adapted using a reinforcement formula in which a *relative similarity measure* is used as a scalar reward or reinforcement signal. The relative similarity measure quantifies how similar the current environment configuration is to the environment configuration encoded by the case relative to how similar the environment has been in previous utilizations of the case. Intuitively, if case matches the current situation better than previous situations it was used in, it is likely that the situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile modifying the case in the direction of the current situation. Alternatively, if the match is not quite as good, the case should not be modified because that will take it away from the regularity it was converging towards. Finally, if the current situation is a very bad fit to the case, it makes more sense to create a new case to represent what is probably a new class of situations.

A detailed description of each step would require more space than is available in this paper (see Ram & Santamaria [1993] for details). Here, we note that 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 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 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 methods presented above have been evaluated using extensive simulations across a variety of different types of environment, performance criteria, and system configurations. We measured the qualitative and quantitative improvement in the navigation performance of the SINS system, and systematically evaluated the effects of various design decisions on the performance of the system. The results show the efficacy of the methods across a wide range of qualitative metrics, such as flexibility of the system and ability to deal with difficult environmental configurations, and quantitative metrics that measure performance, such as the number of navigational problems solved successfully and the optimality of the paths found for these problems. Details of the experiments can be found in Ram & Santamaria [1993].

4 Discussion

Continuous case-based reasoning is a variation of traditional case-based reasoning that can be used to perform continuous tasks. The underlying steps in the method are similar, namely, problem characterization, case retrieval, adaptation and execution, and learning. The ACBARR and SINS systems perform a kind of case-based planning, and in that respect are similar to CHEF [Hammond, 1989]. However, there are several interesting differences due to the continuous nature of the domain, and to the on-line nature of the performance and learning tasks. Our approach is also similar to Kopeikina, Brandau, & Lemmon's [1988] use of case-based reasoning for real-time control. Their system, though not intended for robotics, is designed to handle the special issues of time-constrained processing and the need to represent cases that evolve over time. They suggest a system that performs the learning task in batch mode during off peak hours. In contrast, our approach combines the learning capabilities of case-based reasoning with the on-line, real-time aspects of reactive control. In this respect, our research is also different from earlier attempts to combine reactive control with other types of higher-level reasoning systems (e.g., [Chien, Gervasio, & DeJong, 1991; Mitchell, 1990]), which typically require the system to "stop and think." In our systems, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of Grefenstette & Ramsey [1992].

Our approach combines continuous representations, continuous performance, and continuous learning into a single integrated framework. There are three basic assumptions that underlie this approach. First, the interaction between the reasoning system and the environment is causal and consistent. By *causal* we mean that the same actions executed under the same environmental conditions would result in the same outcomes (or similar outcomes, but such variations are much slower than the execution cycle of the system). By *consistent* we mean that small changes in the executed actions under the same environmental conditions would result in small changes in the outcomes. This guarantees that the system can use past experience to guide performance in similar situations in the future and hope to obtain the same results from its actions. The second assumption underlying our approach is that the system's experiences are likely to be *typical* of future experiences, and that the system will usually encounter problem situations that are similar to those that it already has experience with. These assumptions are also common to many traditional case-based reasoning systems (see, for example, the "typicality assumption" of Ram [1993b]), although they are not always stated explicitly or in this exact manner.

The third assumption is, perhaps, unique to continuous case-based reasoning. In our current work, we have assumed that the problem domain can be represented quantitatively. This is required by the semantic concepts and operations in our method. In particular, our method requires a formal and well-defined similarity metric to judge the similarity of two situations, which is used to determine the direction and magnitude of the necessary adaptations. While case-based reasoning in non-continuous domains also requires a similarity metric for partial matching, most such metrics used in existing systems are not fine-grained enough to determine the degree of similarity between continuous cases or to place situations that could vary continuously and infinitesimally from each other on a similarity scale. This assumption could be relaxed if adequate symbolic representations and similarity metrics could be developed, but more research is needed into this issue.

Our approach to continuous case-based reasoning introduces several innovations to the basic case-based reasoning paradigm. Due to the similarity in the assumptions and methods, we conjecture that many of these innovations would be useful in traditional case-based reasoning systems as well. Let us discuss the underlying case-based reasoning issues in more detail.

4.1 Continuous cases

Our methods represent a novel approach to case-based reasoning for a new kind of task, one that requires continuous, on-line performance. The system must continuously evaluate its performance, and continue to adapt the solution or seek a new one based on the current environment. Furthermore, this evaluation must be done using the simple perceptual features available to a reactive control system, unlike the complex thematic features or abstract world models used to retrieve cases and adaptation strategies in many case-based reasoning systems. Case-based reasoning in such task domains requires a continuous representation of cases, in terms of the available features, that represent the time course of these features over suitably chosen time windows. These representations are learned and modified continuously while simultaneously being used to guide action.

4.2 Abstract cases

In a traditional, symbolic task domain, a case represents an actual experience or an abstraction of one. For example, cases in CHEF [Hammond, 1989] represent actual recipes created by the program. In a continuous domain, however, an actual experience consists of the time histories of real parametric values of perceptual and control parameters. For example, a navigational experience might involve a starting location of (2.34 m, 1.23 m) on a two-dimensional grid, and a destination location of (6.71 m, 10.98 m) on that same grid. The experience might consist of the values of perceptual parameters such as the current position of the robot (an x-y coordinate pair of real numbers), number of obstacles sensed in the immediate vicinity of the robot (an integer), or the distance in meters to the nearest obstacle, and control parameters such as the minimum safe distance of approach to an obstacle (in meters), the speed of the robot (in meters per second), or the current heading of the robot (in radians). These parameters vary continuously over time; thus the complete experience is represented by time graphs of each of these parameters. Clearly, it is not feasible to store the entire time history of each of these parameters for each problem that the robot solves, nor is it useful to do so.[2]

Thus SINS learns and stores some abstraction of the actual experience. One might argue that CHEF actually does the same, since an actual cooking experience would involve perceptual input (such as looking at the frying pan and judging whether the ingredients have browned enough) and control output (such as moving the spatula to stir the ingredients in the pot). In CHEF, the experiences have already been abstracted by the programmer and represented in symbolic form. One of the open issues in our research is the automatic extraction of such symbolic representations from the actual continuous experiences of the system (e.g., [Kuipers & Byun, 1988]).

4.3 Virtual cases

A related, but different, problem with continuous task domains is that an experience can differ from another in infinitely many ways, and differences can range from significant to infinitesimal. Only a tiny fraction of all possible experiences will actually be undergone by the system. However, the power of a case-based reasoning system comes from its cases, and so it is desirable to have a representative library of cases that will cover the range of experiences the system is likely to encounter. We introduce the idea of a *virtual case*, which represents a representative experience that the system could well have had but may or may not actually have had. Rather than trying to remember all the details (down to the grain-size defined by the programmer) of each experience, or some abstraction of these details, SINS combines past cases and present experiences to create a virtual experience. This is similar to the abstraction process described earlier, with the difference that a virtual case does not represent an abstract or generalized description of an actual experience but rather a virtual experience derived from a combination of several actual experiences. The notion of a virtual case might be useful in symbolic case-based reasoning systems as well. To take a simple example, AQUA learns about and updates existing cases based on new experiences [Ram, 1993b]. This process results in hypotheses that may or may not be "true" of a single experience, but are useful and plausible. If used in future reasoning, these hypotheses could be viewed as virtual cases. SINS's virtual cases are more sophisticated since they are continuously refined through use; the refinement algorithms are also different, as discussed below.

4.4 Two types of behavior modification

As discussed earlier, our systems use case-based reasoning to suggest global modifications (behavior selection) as well as to suggest more local modifications (behavior adaptation). The knowledge required for both kinds of suggestions are stored in a case, in contrast with traditional case-based reasoning systems in which cases are used only to suggest solutions, and a separate library of adaptation rules is used to adapt a solution to fit the current problem. In many problem domains, even non-continuous ones, planning (or other types of problem-solving) cannot be performed separately from plan execution. In such situations, case-based reasoning can be used to propose a plan (or solution) as well as continuously refine it during execution. Another difference of interest is that cases in ACBARR and SINS propose modifications, not directly of the plan or trajectory, but of the reactive planner itself which then result in modifications to the proposed trajectories.

4.5 Two types of adaptation

Traditional case-based reasoning systems retrieve cases and adapt the solutions proposed by those cases in order to provide new solutions to new problems at hand. However, in order to build virtual cases, our systems also need to adapt the cases themselves in response to new experiences. This is similar to the incremental case modification process in AQUA [Ram, 1993b] in that, in addition to using a case to deal with a new situation, the system can use an experience to learn more about its existing cases.

In our problem domain, cases represent environmental regularities that have been identified by the system through its experience. These provide the predictive power necessary to navigate in future situations. Cases are used for behavior adaptation;

[2]However, if the range of allowable variations of perceptual and control parameters is bounded by the nature of the task, such a "memory-based approach" may be in fact be feasible [Atkeson, 1990].

in standard case-based reasoning terms, this can be viewed as the process of using the recommendations provided by the case to adapt the solutions currently being pursued by the system. In addition, cases themselves can be adapted through experience; this can be viewed as a process of discovery in which the system develops a model of the world around it. The system "explores" the search space as its case representations traverse this space and find good "niches" representing regularities. The former process could be viewed as a process of "solution adaptation," and the latter as one of "case modification."

The particular method of case modification used in our system is similar to that of Sutton [1990], whose system uses a trial-and-error reinforcement learning strategy to develop a world model and to plan optimal routes using the evolving world model. Unlike this system, however, our system does not need to be trained on the same world many times, nor are the results of its learning specific to a particular world, initial location, or destination location. In general, we hypothesize that case modification may be useful in other types of case-based reasoning systems as well, although different methods may need to be developed to perform the modifications. AQUA's incremental case modification, for example, can be viewed as such an extension to SWALE's solution adaptation [Schank, 1986].

Different criteria may also need to be developed for deciding when to modify a case to fit the new experience, when to learn a new case to represent the new experience, and when to use the case for solution adaptation but without modifying it. Our system uses a relative similarity measure to identify potential regularities. Intuitively, if case matches the current situation better than previous situations it was used in, it is likely that the situation involves the very regularities that the case is beginning to capture; thus, it is worthwhile modifying the case in the direction of the current situation. Alternatively, if the match is not quite as good, the case should not be modified because that will take it away from the regularity it was converging towards. Finally, if the current situation is a very bad fit to the case, it makes more sense to create a new case to represent what is probably a new class of situations.

4.6 On-line real-time response

Unlike traditional case-based reasoning systems which rely on deep reasoning and analysis (e.g., [Hammond, 1989]), and unlike other machine learning augmentations to reactive control systems which fall back on non-reactive reasoning (e.g., [Chien, Gervasio, & DeJong, 1991]), our method allows the system to continue to perform reactively with very little performance overhead as compared to a "pure" reactive control system. Even if real-time response is not required, however, continuous case-based reasoning could still be used in problem domains which are inherently continuous and require continuous representations.

4.7 Adaptive reactive control

Our research also contributes to reactive control for autonomous robots in the following ways. One, we propose a method for the use of assemblages of behaviors tailored to particular environmental demands, rather than of single or multiple independent behaviors. Two, our systems can select and adapt these behaviors dynamically without relying on the user to manually program the correct behavioral parameters for each navigation problem. Three, the knowledge required for behavior selection and modification is automatically acquired through experience using multiple learning methods. Finally, our system exhibits considerable flexibility over multiple domains. For example, it performs well in uncluttered worlds, highly cluttered worlds, worlds with box canyons, and so on, without any reconfiguration. In this paper, we have focussed on the case-based reasoning aspects of our work; robot control issues are discussed in [Ram, Arkin, Moorman, & Clark, 1992; Ram & Santamaria, 1993].

5 Conclusions

We have presented a novel method for augmenting the performance of a reactive control system that combines case-based reasoning for on-line parameter adaptation and reinforcement learning for on-line case learning and adaptation. The method is fully implemented and has been evaluated through extensive simulations.

The power of the method derives from its ability to capture common environmental configurations, and regularities in the interaction between the environment and the system, through an on-line, adaptive process. The method adds considerably to the performance and flexibility of the underlying reactive control system because it allows the system to select and utilize different behaviors (i.e., different sets of schema parameter values) as appropriate for the particular situation at hand. SINS can be characterized as performing a kind of constructive representational change in which it constructs higher-level representations (cases) of system-environment interactions from low-level sensorimotor representations [Ram, 1993a].

In SINS, the perception-action task and the adaptation-learning task are integrated in a tightly knit cycle, similar to the "anytime learning" approach of [Grefenstette & Ramsey, 1992]. Perception and action are required so that the system can explore its environment and detect regularities; they also, of course, form the basis of the underlying performance task, that of navigation. Adaptation and learning are required to generalize these regularities and provide predictive suggestions based on prior experience. Both tasks occur simultaneously, progressively improving the performance of the system while allowing it to carry out its performance task without needing to "stop and think."

There are still several unresolved issues in our research. While we have been able to determine appropriate time windows for SINS through simulation studies, the size or extent of the cases needed to represent extended experiences in continuous domains is still an open issue [Kolodner, 1993]. Furthermore, the retrieval process is very expensive and limits the number of cases that the system can handle without deteriorating the overall navigational performance, leading to a kind of utility problem [Minton, 1988]. Our current solution to this problem is to place an upper bound on the number of cases allowed in the system. A better solution would be to develop a method for organization of cases in memory; however, conventional memory organization schemes used in case-based reasoning systems (see [Kolodner, 1993]) assume structured, nominal information rather than continuous, time-varying, analog information of the kind used in our cases.

Another open issue is that of the nature of the regularities captured in the system's cases. While SINS' cases do enhance its performance, they are not easy to interpret. Interpretation is desirable, not only for the purpose of obtaining of a deeper understanding of the methods, but also for possible integration of higher-level reasoning and learning methods into the system. For example, instead of guessing initial schema parameter values or modifying them incrementally through trial and error, an explanation-based module working on top of the case adaptation module could provide better suggestions for these values, thus speeding up the search process of finding the best schema parameter values associated with a particular environment situation. This requires a symbolic understanding of the knowledge represented in the system's cases.

Despite these limitations, SINS is a complete and autonomous self-improving navigation system, which can interact with its environment without user input and without any preprogrammed "domain knowledge" other than that implicit in its reactive control schemas. As it performs its task, it builds a library of experiences that help it enhance 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.

References

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

Ashley, K. & Rissland, E. [1987]. Compare and contrast, a test of expertise. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 273–284.

Atkeson, C.G. [1990]. Memory-based learning in intelligent control systems. In *Proceedings of the American Control Conference*, San Diego, CA.

Brooks, R. [1986]. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23.

Chien, S.A., Gervasio, M.T., & DeJong, G.F. [1991]. On becoming decreasingly reactive: Learning to deliberate minimally. In *Proceedings of the Eighth International Workshop on Machine Learning*, 288–292, Chicago, IL.

Clark, R.J., Arkin, R.C. & Ram, A. [1992]. Learning momentum: On-line performance enhancement for reactive systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 111–116, Nice, France, 1992.

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

Georgeff, M. [1987]. Planning. *Annual Review of Computer Science*, 2:359–400.

Grefenstette, J.J. & Ramsey, C.L. [1992]. An approach to anytime learning. In D. Sleeman & P. Edwards (eds.), *Machine Learning: Proceedings of the Ninth International Conference*, 189–195, Aberdeen, Scotland.

Hammond, K.J. [1989]. *Case-Based Planning: Viewing Planning as a Memory Task*. Perspectives in Artificial Intelligence. Academic Press, Boston, MA, 1989.

Kaelbling, L. [1986]. An architecture for intelligent reactive systems. Technical Note 400, SRI International, October.

Kolodner, J.L., Simpson, R.L., & Sycara, K. [1985]. A process model of case-based reasoning in problem solving. In A. Joshi, editor, *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 284–290, Los Angeles, CA.

Kolodner, J.L. [1993]. *Case-based reasoning*. Morgan Kaufmann, San Mateo, CA, in press.

Kopeikina, L., Brandau, R., & Lemmon, A. [1988]. Case-based reasoning for continuous control. In *Proceedings of a Workshop on Case-Based Reasoning*, 250–259, Clearwater Beach, FL.

Kuipers, B.J. & Byun, Y-T. [1988]. A robust, qualitative method for robot spatial learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 774–779, St. Paul, MN.

Maes, P. [1990]. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70.

Minton, S. [1988]. *Learning effective search control knowledge: An explanation-based approach*. PhD thesis, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA. Technical Report CMU-CS-88-133.

Mitchell, T.M. [1990]. Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1051–1058, Boston, MA.

Moorman, K. & Ram, A. [1992]. A case-based approach to reactive control for autonomous robots. In *Proceedings of the AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, Cambridge, MA.

Mostow, J. & Bhatnagar, N. [1987]. FAILSAFE—A floor planner that uses EBG to learn from its failures. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 249–255, Milan, Italy.

Payton, D. [1986]. An architecture for reflexive autonomous vehicle control. In *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 1838–1845.

Ram, A. & Santamaria, J.C. [1993]. A multistrategy case-based and reinforcement learning approach to self-improving reactive control systems for autonomous robotic navigation. In R.S. Michalski & G. Tecuci (eds.), *Proceedings of the Second International Workshop on Multistrategy Learning*, Harpers Ferry, WV.

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 robotic systems. Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology, Atlanta, GA.

Ram, A. [1993a]. Creative conceptual change. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, CO.

Ram, A. [1993b]. Indexing, elaboration & refinement: Incremental learning of explanatory cases. *Machine Learning*, 10:201–248, in press.

Sacerdoti, E.D. [1977]. *A structure for plans and behavior*. Elsevier, New York.

Schank, R.C. [1986]. *Explanation Patterns: Understanding Mechanically and Creatively*. Lawrence Erlbaum, Hillsdale, NJ.

Segre, A.M. [1988]. *Machine Learning of Robot Assembly Plans*. Kluwer Academic, Norwell, MA.

Sutton, R.S. [1990]. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, 216–224, Austin, TX.

Sutton, R.S., [1992], editor. *Machine Learning*, 8(3/4), Special issue on reinforcement learning.

Veloso, M. & Carbonell, J.G. [1991]. Automating case generation, storage, and retrieval in Prodigy. In R.S. Michalski & G. Tecuci (eds.), *Proceedings of the First International Workshop On Multistrategy Learning*, 363–377, Harpers Ferry, WV.