Forgetting Bad Behavior: Memory Management for Case-Based Navigation

Zsolt Kira, Ronald C. Arkin
Mobile Robot Laboratory
College of Computing, Georgia Institute of Technology
Atlanta, Georgia 30332–0250
{zkira, arkin}@cc.gatech.edu

***Abstract*— In this paper, we present successful strategies for forgetting cases in a Case-Based Reasoning (CBR) system applied to autonomous robot navigation. This extends previous work that involved a CBR architecture which indexes cases by the spatio-temporal characteristics of the sensor data, and outputs or selects parameters of behaviors in a behavior-based robot architecture. In such a system, the removal of cases can be applied when a new situation unlike any current case in the library is encountered, but the library is full. Various strategies of determining which cases to remove are proposed, including metrics such as how frequently a case is used and a novel spreading activation mechanism. Experimental results show that such mechanisms can increase the performance of the system significantly and allow it to essentially forget old environments in which it was trained in favor of new environments it is currently encountering. The performance of this new system is better than both a purely reactive behavior-based system as well as the CBR module that did not forget cases. Furthermore, such forgetting mechanisms can be useful even when there is no major environmental shift during training, since some cases can potentially be harmful or rarely used. The relationship between the forgetting mechanism and the case library size is also discussed.**

Keywords: Mobile Robot Navigation, Case-Based Reasoning, Behavior-Based Robotics

I. INTRODUCTION

Behavior-based robotics, which advocates a tight coupling between perception and action, provides good performance in dynamic and unpredictable environments. In this paradigm, the robot chooses a subset of behaviors (a behavioral assemblage) to execute from a predefined behavioral repertoire. The behavioral response is based solely on current sensor data. Usually, the behaviors are parameterized so that they can be applied in many situations, and one problem is that these parameters have to be optimized manually in order to achieve good performance. This optimization requires both knowledge of the robot behaviors and preliminary experimentation. Furthermore, constant parameters may not be optimal for environments that can vary widely.

In order to address these issues, Likhachev et al. have added a Case-Based Reasoning (CBR) module to the architecture [1][2]. Case-Based Reasoning is a method where previous problems or situations, in addition to their solutions, are remembered [3]. When the agent encounters a similar situation again, it can adapt the previous solution to the current one and use it (accommodating for the differences between the two situations). The previous work applied Case-Based Reasoning in order to learn which behavioral parameters to use in given situations. At first the case library was created manually, and later work extended this by adding the capability of learning the cases from scratch. This enhanced system was able to outperform the behavioral system without learning, in which the parameters are held constant throughout the environment. However, one problem that arose is that the case library filled up relatively quickly. Once the library was full, the best matching case to the current environment was adapted, even if it was drastically different. The adaptation changes both the input and output values of the case, and if this adaptation is done on a case that is tuned to an entirely different situation, degraded performance can result. Ideally, the robot should adapt its knowledge as the world changes by creating new cases and removing old ones.

In that system the parameters that each case outputs are optimized using a hill-climbing type strategy. However, the success rates of some cases do not converge, or converge to the lowest possible success rate, and hence do not provide good knowledge that can improve performance. Despite this, they were previously kept in the library and took up space which better cases could have occupied. These cases could also potentially be harmful if they are the most similar cases to the current

This research is supported under DARPA's Mobile Autonomous Robotic Software Program under contract #DASG60-99-C-0081. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

situation in the library and are selected for use. Even though this might be unlikely since the decision to use a case is weighted by success, the alternative of not using them might be even worse because cases which are very different will be chosen and adapted.

The new work reported in this paper extends the system by adding a mechanism by which such cases are removed when the case library is full and a new situation is encountered. The general hypothesis is that a forgetting mechanism, in which cases are deleted based on the strategies proposed here, can improve the performance of a Case-Base Reasoning system. The methods used here are not necessarily restricted to this particular system, although some potential mechanisms for case removal were discarded due to the peculiarities of the applied system (some are mentioned later in the paper). For example, how often a case is used or how recently it has been used are features of cases that can be determined in most Case-Based Reasoning systems. Forgetting can be done by restricting the size of the case library and deleting cases when the case library is full, or it can be done periodically. In this paper we take the former approach, although some investigation into periodic forgetting has been done.

The importance of forgetting has been recognized by other researchers, both in the machine learning community and also specifically with respect to Case-Based Reasoning [4][5]. Markovitch showed that in some domains in machine learning, acquired knowledge can have negative effects even when it is correct [4]. He also showed that randomly forgetting knowledge can improve performance, and that this improved performance cannot be equaled by not learning as much in the first place. What is more striking is that this is true when the removal of knowledge is performed before testing without additional learning taking place. In other words, correct knowledge is randomly removed before testing, and this results in better performance. Furthermore, heuristics determining what to delete can achieve an even greater performance boost over random deletion. Unlike our approach, forgetting in this research occurred after the training phase, and different rates of random deletion were tested.

Watanabe restricted the size of the case library in a CBR planner used to manage power grids [6]. Several properties of the cases were used in deciding what to remove, including the age of the case, its reliability, and its similarity to the new situation. Forgetting cases seemed to be advantageous over a system without forgetting, especially in a changing environment. In other work, Lieber studied criteria for comparing two case libraries, which has implications in forgetting since it can be used to determine the effects of removing a case [7].

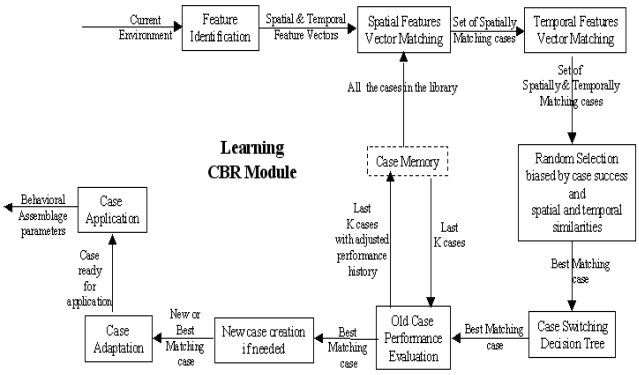

Fig. 1. High-level structure of the learning CBR module. [2]

Of course, deletion can also have adverse effects on competence, or the range of problems covered by the library, if done improperly. Smyth considered a forgetting mechanism that took into account different categories of cases (e.g. *pivotal* cases which are important for the competence of the system) when deleting cases [5]. When compared to random or utility-based deletion, this scheme did not degrade competence as much. Competence preservation and other issues will not be studied here. In the case of autonomous robot navigation, the problem space is not constant as the robot moves to new environments, and hence competence preservation is not necessarily an objective.

II. Architecture

Our previous Case-Based Reasoning architecture for robot navigation can be seen in Figure 1 [2]. The input to the entire system is obtained from the environment in the form of raw sensor data and the goal data. The Feature Identification module analyzes this data and outputs a feature vector for both spatial and temporal characteristics of the sensor data. In order to calculate the spatial feature vector, the space around the robot is split into k uniform angular regions (in the current implementation k is 4). The spatial characteristics involve the *traversability* of each region, which changes as the size of the largest obstacle cluster and the distance to this cluster change. The temporal feature vector involves normalized short-term and long-term robot velocities.

Once these feature vectors are obtained, the best matching case is found through multiple stages of selection. First, the *best spatially matching* case to the current environment is retrieved, where similarity is the weighted Euclidean distance between the spatial feature vectors of the environment and the case. A set of cases is then selected randomly, where the probability of a case being selected is taken from a Gaussian distribution and is proportional to the difference between its spatial

similarity to the environment and the spatial similarity of the best spatially matching case to the environment. Out of this new set of cases, a similar method is used to find the cases that are temporally matching. Finally, one case is selected from the new set randomly, where the probability of a case being selected is proportional to a weighted sum of the case spatial similarity, temporal similarity, and case success. Case success is measured in terms of relative motion to the goal, and is interwoven with a delayed reward mechanism. Refer to [2] for more details.

After a single case has been chosen, a decision tree is used in determining whether that case will replace the case that is currently in use. The decision is based on how long the current case has been active, the spatial similarities, short-term relative motion, and long-term relative motion (full descriptions appear in [2]). If it is decided that a new case is to be used, another decision is made whether a new case should be created based on the temporal and spatial similarities and the success of the selected case. This is where forgetting is added, since in the previous system no new cases are created if the library is full. In the new system, if it is decided that a new case should be created and the library is full, forgetting is applied. If there is no good candidate to remove, then a new case is not created. If there is a good candidate, it is removed and a new case is created in its place.

Finally, there is an adaptation step in which the output parameters of the cases are adapted according to a method similar to hill-climbing. If the case improves due to previous adaptations, the case is adapted in a similar direction once again. If not, then the adaptation direction is changed. There is also a second adaptation step where the output parameters are altered based on the short-term and long-term relative velocities of the robot. Once completed, the case is applied by changing the parameters of the current behavioral assemblage to the ones specified in the new case.

This complete system was integrated into the *MissionLab* system [8], which implements the Autonomous Robot Architecture (AuRA) [9]. The CBR module was integrated into the **GOTO** behavioral assemblage, as depicted in Figure 2. The module's input is sensor and goal data, and the output is the behavioral parameters to be used. The changeable parameters of the assemblage include how much noise to add during navigation and the sphere of influence of the obstacles.

III. Forgetting Strategies

The forgetting mechanism that was implemented resides in the "New case creation if needed" step in Figure 1. Various strategies are now presented for selecting which case should be forgotten when the library is full and the decision to create a new case is made. Three strategies of forgetting have been employed. One interesting idea that was ruled out is to use clustering algorithms to compact the existing set of cases into a few representative cases. In this work, however, this method is not applicable due to the small size of the case library. The strategies that were chosen are now described.

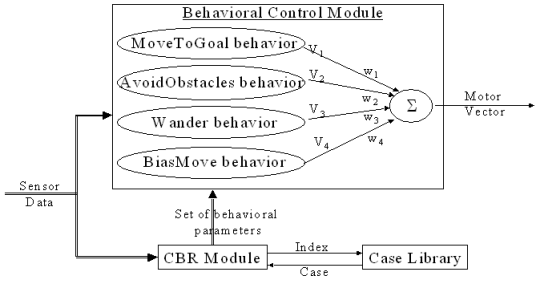

Fig. 2. Interaction between behavioral control module running a **GOTO** behavioral assemblage and CBR unit. [2]

A. Random Forgetting

The simplest forgetting strategy is to pick a random case to remove. Surprisingly, this can improve performance in some domains [4]. Here, however, it was not anticipated to improve performance due to many factors specific to the domain. Since the case library is relatively small and cases are improved through experience, the chances of removing successful cases is very high. Also, there are usually few redundant successful cases in the library. Blindly removing cases will clearly result in very few cases that are well trained, especially in a rich environment in which the robot encounters many new situations. Hence, this strategy is used as a baseline with which other strategies are compared.

B. Metrics for Removal

Several heuristics or metrics can be used to estimate which cases should be deleted. The metrics used include performance, recency, and relative use frequency:

$$P(C) = \frac{S(C)+1}{2} \tag{1}$$

where $P(C)$ is the performance of case C and $S(C)$ is the success as defined in the previous work. Note that $S(C)$, the success of case C, is bounded between -1 and 1.

$$R(C) = \frac{1}{t_e^C + 1} \tag{2}$$

where $R(C)$ is the recency of case C and t_e^C is the time elapsed since the last application of case C.

Pseudocode for Weighted Metrics Forgetting

If new case should be created and library is full
  Case to be deleted = $argmin_{c \in L}(WeightedMetric(c))$
  where WeightedMetric(c) = $(w_1 * P(c) + w_2 * R(c) + w_3 * U(c))$
  If WeightedMetric(Case) $< threshold_{WM}$ then
    Remove Case and create new one in its place
  Else
    Do not remove, and do not add a new case

TABLE I
PSEUDOCODE FOR THE WEIGHTED METRICS FORGETTING STRATEGY.

$$U(C) = \frac{t_a^C}{\sum_{C' \in L} t_a^{C'}} \tag{3}$$

where $U(C)$ is the relative use frequency of case C, t_a^C is the number of time steps case C has been applied, and the denominator is the summation of time steps that all current cases in the case library (L) have been applied (note that this may be different from the number of time steps the system has been running since cases may be removed). The division serves to normalize $U(C)$ so that it is bound between 0 and 1.

For all of these metrics, higher numbers indicate better cases which should be less likely to be deleted. Some functional combination of these metrics (e.g. a weighted sum) can be used to determine which case to replace when a new situation that is dissimilar to any other case appears and the case library is full. One problem is determining the function or relative weights of the metrics. The method used here is to measure the performance gains of the metrics independently and use this data to suggest good weights. Although there can clearly be a complex interaction between the strategies, this is a relatively reasonable and objective approximation. Table I shows the process by which the worst case is selected for deletion.

C. Activation for Forgetting

The final strategy evaluated is based on spreading activation. Each case C has an activation level A_C associated with it, and a case is activated upon certain events. When a case is first created, it is activated by a constant α:

$$A_C = \alpha \tag{4}$$

Subsequently, when a case is selected for use it is activated. Its activation level is proportional to its similarity to the current environment, $S_{C,E}$:

$$A_C = A_C + S_{C,E} * \lambda, \qquad 0 \leq \lambda \leq 1 \tag{5}$$

When the success S(C) of a case is changed, it is activated as well by an amount proportional to the change (which can be negative):

$$A_C = A_C + \triangle S(C) * \sigma, \qquad 0 \leq \sigma \leq 1 \tag{6}$$

The activation level of all cases decreases as time passes:

$$A_C = A_C * \gamma, \qquad 0 \leq \gamma \leq 1 \tag{7}$$

This strategy accounts for some of the metrics used in the previously described strategy. For instance, activating a case when used accounts for recency, and success is accounted for by the change in activation corresponding to change in success. There are some differences, however. The first difference is that some history is automatically maintained; previous activations of the case will remain for several time steps (how long depends on constants such as decay rate). When using the metrics alone, only the values of the metrics in the current time step are used. The second difference lies in the spreading activation. When a case is activated, cases that are similar to it (in terms of input space) are activated as well but with a decayed amount. The amount is also proportional to its similarity with the initially activated case. This is computed as follows (suppose Case C has just been activated by amount A):

$$A_{C'} = A_{C'} + A * \mu * S_{C,C'}, \qquad (\forall C' \neq C) \in L \qquad 0 \leq \mu \leq 1 \tag{8}$$

The justification is that usually, environments tend to change slowly. Hence cases that are similar to the currently used case should be kept in the event that the situation changes slightly. The deletion policy is based on the activation level of the cases: The case that is least activated and whose activation is below a threshold is selected for deletion. The threshold prevents cases from being deleted if all cases happen to be highly activated. The pseudocode for this strategy is similar to Table I except that instead of $WeightedMetrics(Case)$, A_{Case} (the activation of the case) is used.

IV. EXPERIMENTAL RESULTS

The modified system with the various forgetting strategies was implemented and tested in *MissionLab*'s simulation environment. For all experiments the case library was of size seven, and navigation was conducted using a modified **GOTO** behavior containing Swirl, where obstacle avoidance vectors circulate around the obstacles [10]. This usually leads to better performance and hence would favor a purely reactive system. All training was done twice to obtain two different case libraries, and all results are averages of these libraries. There were some differences in the performance between the case libraries for certain strategies, mainly

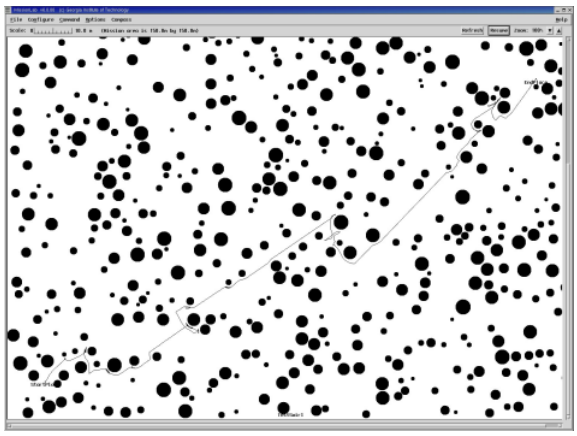

Fig. 3. Homogeneous environment in which the case library was trained and tested. The environment shown here is with a fifteen percent density. Other densities were used for training and testing as well.

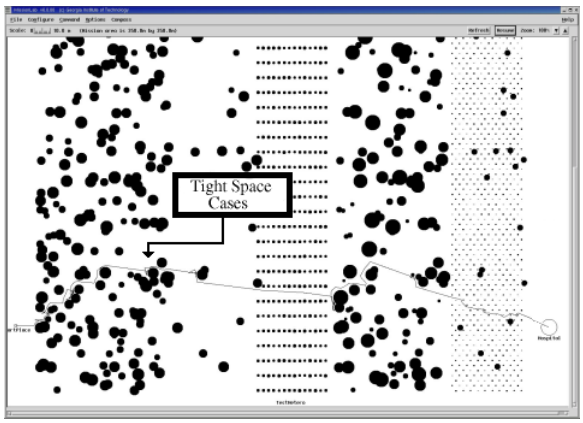

Fig. 4. Heterogeneous-1 environment, with several different phases in which the obstacle patterns are changed.

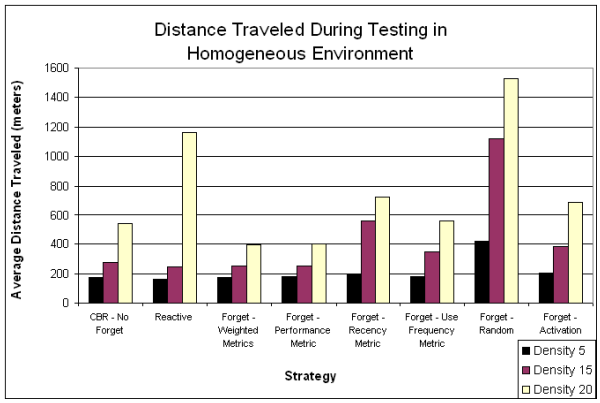

Fig. 5. This graph shows the distance traveled by the robots during the missions. Three obstacle densities were tested for the homogeneous maps. Notice the *performance* and *weighted* metrics outperformed both the purely reactive system as well as the original CBR architecture for higher densities.

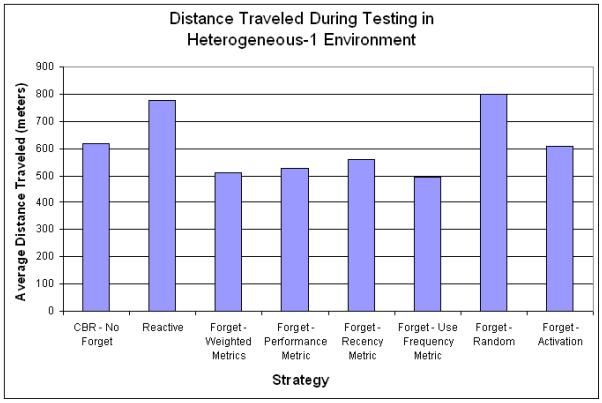

Fig. 6. Distance traveled during missions in the Heterogeneous-1 environment. Once again, the *performance* and *weighted* metrics performed extremely well, with the *use frequency* and *recency* metrics performing comparably. The purely reactive version only finished 78 percent of missions and hence did not perform very well.

in the random and *recency* metric strategies. All stated results are statistically significant, and in fact the more successful strategies resulted in the additional benefit of smaller standard deviations. The thresholds for deletion were chosen to be 40% of the maximum values (the values were not optimized in any way). The metrics used to judge performance were the average distance traveled to complete the missions (in meters), as well as the percentage of missions completed.

The first environment the simulated robot was trained in consisted of a homogeneous environment with randomly placed obstacles in a particular density, as shown in Figure 3. Five densities were used for training: 0, 5, 10, 15, 20, and 25 percent clutter. During training, two versions (with different random seeds) for each density was used, and these maps were used for training ten times. The system was then trained on six empty maps without obstacles, consistent with previous work. The complete training cycle was repeated twice, giving a total number of 242 training missions.

The resulting case libraries for each strategy were then frozen. No learning of new cases or fine tuning of existing cases occurred during evaluation. The libraries were tested on both the homogeneous environments as well as heterogeneous environments. Each of the three densities for the homogeneous environment was tested on 100 maps (two random seeds 50 times each). The average distance traveled during these tests can be seen in Figure 5. In the easier maps with lower obstacle densities, forgetting did not have an impact on performance and in fact reduced it by a small amount. However, with a 20 percent density, the CBR module without forgetting and the purely reactive system both did not scale well and performed poorly. The *weighted metrics*

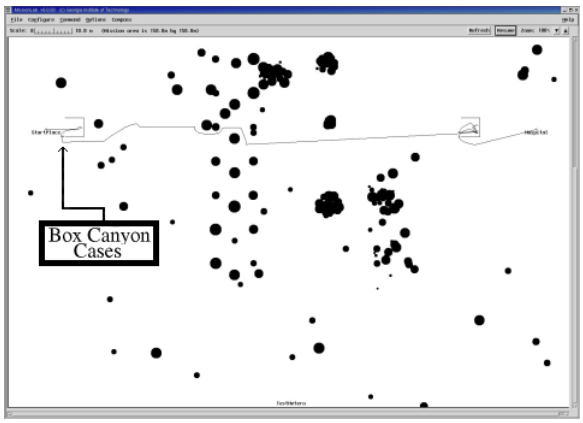

Fig. 7. Heterogeneous-2 environment - A different environment with several different patterns of obstacles, including a box canyon.

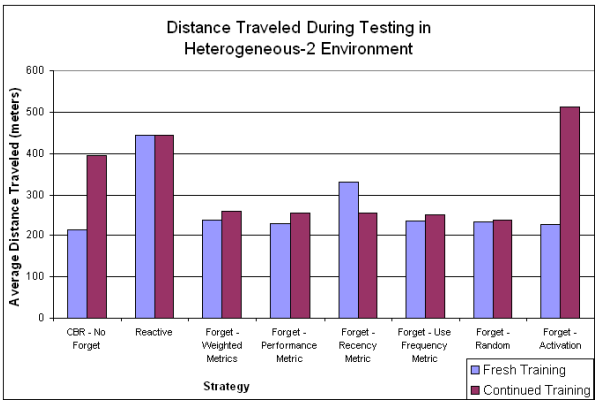

Fig. 8. Distance traveled during missions in the Heterogeneous-2 environment. The "fresh training" version is the case library that was trained on the Heterogeneous-2 environment starting with an empty library. The other started training using the case library that resulted from previous training on the homogeneous environment. Notice the performance for the successful forgetting strategies was almost the same in both versions, whereas the original CBR module degraded in performance when it was re-trained on a new environment.

and *performance* metric were the most successful, with the *use frequency* metric performing almost as well. It is of interest that the *performance* metric had many more cases with a success rate above zero than the *use frequency* metric did (five and a half versus two on average, respectively) yet it performed almost as well. The other strategies did not scale. As expected, the random deletion strategy performed worst. In terms of the number of missions completed, most of the systems using CBR completed in the range of 90 percent of the missions, except the random strategy which completed about half in the harder maps. The reactive version completed all of the missions for the first two densities, but only about 64 percent of the hardest environment with an obstacle density of 20 percent.

The testing on heterogeneous environments was conducted on four different maps with several variations, totaling to 54 runs. The environments, shown in Figure 4, are referred to here as Heterogeneous-1. Figure 6 shows the results of the testing. Once again, the *performance* metric and the *weighted metrics* performed the best among the strategies proposed here, with the *use frequency* and *recency* metrics performing comparably, and slightly beat the system without forgetting. They also completed more missions than the system without forgetting, with a difference of about four percent. The rest performed about as well or worse than the original. In this case, the purely reactive version did fairly badly because it completed much fewer missions. All of the modules using CBR completed most of the missions but the reactive version only completed about 78 percent of them. Furthermore, training was not conducted on this type of environment at all, and the parameters of the reactive version were manually optimized for this environment. This shows that the cases learned in the original homogeneous environment are relatively general and are applicable in other situations. Figure 4 shows a robot run that used the *performance* metric strategy during evaluation. It can be seen that the case library includes cases for tight situations where squeezing through a dense obstacle field is required.

In order to verify that the robot can forget cases which are no longer relevant to its current situation, the case library from the first training session was unfrozen and used for training on a completely different environment (referred to here as Heterogeneous-2). Training was also conducted on these environments starting from an empty library, in order to compare the performance when trained solely on the new environment versus when training uses a previously trained library. Three maps similar to Figure 7 were used for training, in addition to empty environments, with a total of 132 missions. As can be seen, the environment is completely different, with various patterns of obstacles, including a box canyon. After training, the maps were modified in various ways, and testing was conducted on three maps with three variations each, for ten times each yielding a total of 90 testing runs.

As can be seen from Figure 8, when the training used an initially empty library most CBR systems performed about equally well, except for the *recency* metric. All the CBR systems outperformed the reactive-only architecture significantly, since the environment consisted of box canyons which purely reactive architectures typically have trouble addressing. The CBR modules, however, learned useful cases for the box canyon situation and were able to consistently escape from it. When training was performed using the library from the homogeneous

environment training, the advantages of forgetting became apparent. The performance of the system without forgetting decreased by a large amount, but when using the successful forgetting strategies, the performance was almost the same as when using an empty library. There was slight degradation for some of the metrics especially regarding the number of missions completed, but the system with forgetting faired better than without.

Furthermore, many of the forgetting strategies were able to retain successful cases from the original homogeneous environment that were also useful in the new environment. Figure 9 shows the number of cases in each library after training whose success rate was above zero. This is only one of the metrics used, but it shows that most forgetting strategies resulted in more converged cases than the previous architecture. All resulted in libraries with cases for an empty environment in which none of the four regions around the robot are obstructed by obstacles. Some of the more successful strategies had cases for which the robot was surrounded by a particular pattern of obstacles and the noise persistence was set to a low value. Such cases represent parameters that the robot can use to maneuver out of tight spaces.

Cases that were successful in the homogeneous environment but were not useful for the new environment were usually removed, since the various metrics such as case use and case success decreased. This is why some strategies had fewer successful cases after re-training than they started with. However, this also occurred for the other forgetting strategies even if they ended up with more total successful cases, although this cannot be seen from the graph. In many instances unsuccessful cases were replaced by new successful ones. This shows that the forgetting strategies can indeed selectively retain cases from the old environment that are still useful, discard those that are not, and learn new ones in the place of the unsuccessful ones. As seen in Figure 7, the robot using the *performance* metric strategy was able to learn cases for getting out of the box canyon quickly.

Different library sizes were tried in order to ascertain the efficacy of forgetting when the library is large. Sizes of 4, 7, 10, and 20 were tried since the maximum number of cases created by the architecture never reached 20 even with an unlimited case library. Figure 10 shows the results of these experiments. As can be seen, with small sizes forgetting is extremely advantageous and produces significant performance increases. One notable exception is a library of size 4 in the heterogeneous environment. This shows that the performance of the system can be less stable when the library is too small. As the size grows larger, however, the performance of the two systems converged since forgetting never occurred due to the large case library

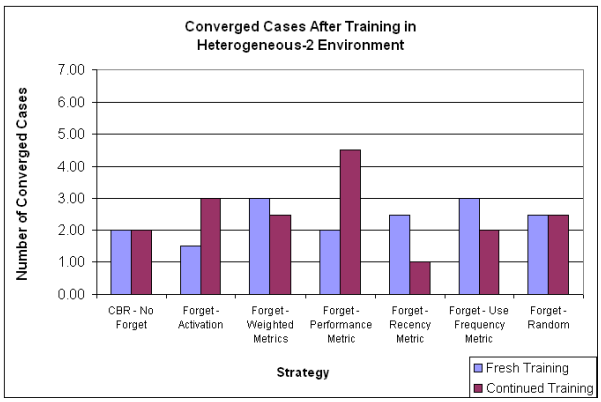

Fig. 9. Number of cases with success rates above zero after training. The *performance* metric uses exactly this criteria in removing cases, so it is not surprising that it has the most converged cases. Other forgetting strategies resulted in more converged cases as well.

size. This shows that in situations in which memory is restricted or the environment is complex, forgetting should be applied. It would be interesting to see if periodically forgetting could improve performance even when memory is unlimited. To that end, the weighted metric mechanism was tested on the homogeneous and heterogeneous environments. It did not result in any performance increase in these environments, although it has not been tested when the robot is trained on one environment and then moved to a drastically new environment. This is where forgetting has been shown to be the most beneficial.

V. Conclusions

The results of this research clearly show that the *performance* and *use frequency* metrics result in the best overall performance, both in terms of distance traveled and missions completed. The *weighted metrics*, in which the weights are proportional to the performance of the individual metrics alone, seemed to inherit the advantages of both of the successful metrics. The spreading activation mechanism did not perform as well and was actually slightly worse than the architecture without forgetting. There are many possible reasons for this. It was originally anticipated that much like the metric strategies, the thresholds and constant parameters could be set to reasonable values and the system would work well. However, in the spreading activation strategy, careful thought has to go into selecting the constant thresholds. In the current system, the activation that is given when a case is used is approximately equal to the activation rates given when other events occur. This should likely not be the case since certain events, such as when a case is being used, occur many orders of magnitude more frequently than others. This means that such events are given more weight, and the parameters

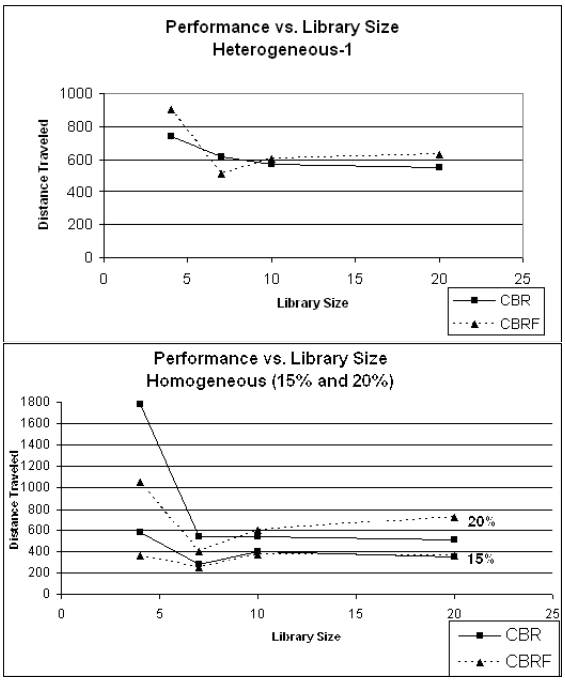

Fig. 10. Performance of forgetting with weighted metrics (CBRF) and non-forgetting systems (CBR) as the case library size is varied.

should be adjusted accordingly. This will be addressed in future work.

The *recency* metric also did not perform as well and exhibited unstable behavior. This may be a result of the large amount of thrashing due to the nature of the metric. In certain areas of the map a few cases are used frequently, whereas in other parts a different set is used. This results in the system deleting and re-creating the same set of cases over and over. Certain threshold parameters can be adjusted to account for this, but it is likely that it is just not a good metric. As expected, the random deletion also performed poorly and was extremely unstable.

There are many implications of these results. First of all, forgetting cases that are poor in terms of the metrics we have defined results in a significant performance increase. This type of forgetting does not have to be restricted to situations where there is an environmental shift during the lifetime of the robot; indeed, significant performance increases were seen while training within the same environment. The usefulness of such a mechanism increases if there is an environmental shift, however. The robot can retain the knowledge it has acquired through experience, but only if it is still useful for its current situation. Another important result that has been shown is that the Case-Based Reasoning system can indeed learn general cases that are useful in many different situations, as shown by the fact that there were cases that continued to prove successful in a drastically different environment. The environments were not engineered for this to happen; it occurred automatically. Finally, it is important to note that many of the metrics used here are not specific to the current navigational system. For example, the *use frequency* metric performed quite well and can be applied to any CBR system. Furthermore, unless all of the cases have optimal knowledge, case success can probably be measured in many systems.

Another implication of these results is that instead of training the robot in an environment and then halting the learning mechanisms, the robot can continue training throughout its lifetime, i.e., lifelong learning is possible. One questions that this work has not addressed is the performance of the robot during training. Even if the performance is not good there are various ways that this can be addressed. For example, the robot can have certain thresholds based on measurements such as how many new situations not reflected in the library it has encountered, and if it is below this threshold it can halt training. It can later restart training when and if it detects that it has entered a drastically new environment. Future work can be done in the area of periodic forgetting and its performance during an environmental shift.

ACKNOWLEDGMENT

The authors would like to thank Michael Kaess for his initial work on the metrics used.

REFERENCES

[1] Likhachev, M. and Arkin, R.C., “Spatio-Temporal Case-Based Reasoning for Behavioral Selection,” Proc. 2001 IEEE International Conference on Robotics and Automation, pp. 1627–1634. 2001.

[2] Likhachev, M., Kaess, M., and Arkin, R.C., “Learning Behavioral Parameterization Using Spatio-Temporal Case-Based Reasoning,” Proc. 2002 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1282–1289, 2002.

[3] Kolodner, J., Case Based Reasoning, Morgan Kaufmann Publishers, San Mateo, 1993.

[4] Markovitch, S. and Scott, P., “The role of forgetting in learning,” In Machine Learning: Proc. of the Fifth Intl. Conf., pp. 459-465. San Mateo: Morgan Kaufmann. 1988.

[6] Watanabe, H., Okuda, K., and Fukiwara, S., “A Strategy for Forgetting Cases by Restricting Memory,” IEICE Transactions on Information and Systems, pp. 1324–1326, October 1995.

[5] Smyth, B. and Keane, M., “Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems,” Proc. 13th International Joint Conference on Artificial Intelligence, pp. 377–382. Montreal: IJCAI, 1995.

[7] Lieber, J., Haton, J.P., Keane, M., and Manago, M., “A criterion of Comparison Between Two Case Bases”, Advances in Case-Based Reasoning, Second European Workshop, pp. 87–100. November 1994.

[8] Mackenzie, D., Arkin, R.C., and Cameron, J, “Multiagent Mission Specification and Execution,” Autonomous Robots, 4(1), pp. 29–57, 1997.

[9] Arkin, R.C. and Balch, T., “AuRA: Principles and Practice in Review,” Journal of Experimental and Theoretical Artificial Intelligence, 9(2), pp. 175–189, 1997.

[10] Arkin, R.C., Behavior-based Robotics, MIT Press, 1998.