# Considerations for Real-Time Spatially-Aware Case-Based Reasoning: A Case Study in Robotic Soccer Imitation

Michael W. Floyd, Alan Davoust, and Babak Esfandiari

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario

**Abstract.** Case-base reasoning in a real-time context requires the system to output the solution to a given problem in a predictable and usually very fast time frame. As the number of cases that can be processed is limited by the real-time constraint, we explore ways of selecting the most important cases and ways of speeding up case comparisons by optimizing the representation of each case. We focus on spatially-aware systems such as mobile robotic applications and the particular challenges in representing the systems' spatial environment. We select and combine techniques for feature selection, clustering and prototyping that are applicable in this particular context and report results from a case study with a simulated RoboCup soccer-playing agent. Our results demonstrate that preprocessing such case bases can significantly improve the imitative ability of an agent.

## 1 Introduction

When using a case-based reasoning (CBR) system, the performance of the system is highly dependant on the quality of the case base that is used [1]. One aspect of case base quality is how well the cases in the case base represent the set of possible problems that the CBR system might encounter. If the case base contains too few cases (or cases that are highly similar to each other) then the case base might not adequately cover the problem space leading to a decrease in performance. One reason, which we will focus on in this paper, for a case base being a less than ideal size is if the CBR system must operate under a real-time constraint. Since the CBR system must search the case base in order to determine the solution to a problem, a larger case base will likely lead to a longer search time.

One specific area where CBR can be applied in a real-time setting is in the imitation of spatially-aware autonomous agents. These agents are able to identify objects that are visible to them in their environment and perform actions based on the configuration of those objects. Unless the agent has a complete world view, it is generally only able to see a subset of objects at any given time. In addition to being able to only view a subset of the objects in the environment at a given time, the agent may also not know the total number of objects that exist in the environment.

When an agent is fully aware of each unique object in the environment and is able to differentiate between similar objects (for example, the agent can differentiate between two humans and does not just classify them both as *human*) then there would exist a *type* for each of the unique objects. However, if the agent is unable to differentiate between similar objects then multiple objects would belong to a single *type* and the objects of a similar type could be considered interchangeable.

The RoboCup Simulation League [2] is a realistic benchmark for examining the type of agents we described. RoboCup agents must deal with temporal events as well as with an environment consisting of a 2-D space (the soccer field) with objects and other agents within that space. The agent does not know its exact position on the field but must estimate it using the location of the objects that are visible to it. During each time period in a game, the server provides clients with world view and state information (subject to a noise model) using one of `see`, `hear`, or `sense_body` messages. Objects described in `see` messages may be players, the ball, goal nets, or the numerous lines and flags located on the field. Due to noise associated with *seeing* objects, a RoboCup agent often does not possess enough information to properly differentiate similar objects (for example, it may only be able to tell that it can see a player, though not which player it sees). The RoboCup Simulation league provides a suitable testbed to examine the methods described in this paper due to the real-time constraints and the difficulty in differentiating similar objects.

In the remainder of this paper we will examine several techniques (feature selection, clustering and prototyping) that can be used to increase the number of cases that can be examined within a real-time limit as well as methods of improving the diversity of cases contained in a fixed sized case base. It should be noted that assume that the case bases do not use any method of fast-indexing, although we feel that our techniques could be used as a complement to fast-indexing. Initially, in Section 2 we describe the case study we will perform to demonstrate the techniques presented in later sections of the paper. Section 3 will look at methods for representing a case and selecting the most useful features in a case to use will be covered in Section 4. The creation of prototype cases will be covered in Section 5. Related work will be examined in Section 6 followed by conclusions in Section 7.

## 2 Case Study: RoboCup Simulation League

Our case study involves a case-based reasoning system that is used to imitate the behaviour of a RoboCup [2] soccer player. Cases for this system are generated, in an automated manner, by observing the RoboCup player that will be imitated and logging the inputs to the player and the player's outputs [3,4]. Each case is comprised of the inputs to the player (what objects the player can see) as well as the outputs (actions the player performs) of the player in response to those inputs. The imitative CBR agent then uses those cases in an attempt to select appropriate actions based on what it can currently see (what objects are in its field of vision). The goal of such a system is to produce behaviour that is indistinguishable from the behaviour of the RoboCup player it is imitating.

In the RoboCup Simulation League, the environment contains objects that belong to a fixed number of *object types*. Although each individual object on the field in unique, the agent is often unable to distinguish between objects of the same type due to noise. For example, the agent would be able to see a teammate but might not be able to tell what specific teammate it is. For this reason, objects of the same type are treated as interchangeable. In the RoboCup Simulation League we define the following object types:

$$Type = \{Ball, Goalnet, Flag, Line, Teammate, Opponent, Unknownplayer\}$$

Each player may only perform an action once per discrete time interval, called a cycle. If the player does not perform an action each cycle then it will be at a disadvantage compared to other players who act more often. The entire process (Figure 1) of identifying what objects are currently visible to the agent, using CBR to select the appropriate actions to perform and performing those actions should then be completed within a cycle (of length 100ms). Also, given that the CBR process is not the only task the agent needs to complete within each cycle, we will set our time limit for performing CBR to half of the cycle (50ms).

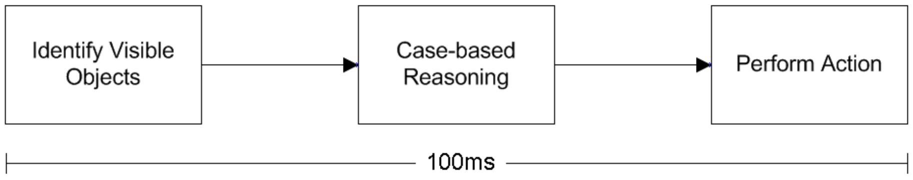

**Fig. 1.** The activities the agent must perform in one 100ms cycle

## 2.1 Metrics

The effectiveness of each approach will be measured using a combination of two criteria: how much time it takes to perform the case-based reasoning process and how well the agent performs imitation.

The time it takes to perform the CBR process will be measured as the time it takes from when the CBR system is given a problem (the objects an agent can currently see) to when it provides a solution (the action to perform). As was mentioned previously, we want this time to be as close to our imposed time limit of 50ms as possible. If the time is lower than 50ms we could add more cases to our case base, or if the time is greater than 50ms we would need to remove cases from the case base. Either adding or removing cases may have an impact of how well the imitative agent performs, so we also require a metric of agent performance.

If it takes an amount of time, $T$, to perform CBR using a case base of size $N$, then we can estimate the number of cases, $N_{max}$, that can be used within our real-time limit, $T_{max}$, as:

$$N_{max} \approx \frac{T_{max} * N}{T} \tag{1}$$

A simple measure of agent performance would be to measure the classification accuracy of the CBR system when performing a validation process. Each case in a testing set will be used as input to the CBR system and the output of the CBR system, the predicted action, will be compared to that case's known action. This provides a measure of the number of test cases that are classified correctly. However it can be misleading when the testing data contains a disproportional number of cases of a certain class. For example, in RoboCup soccer a player tends to *dash* considerably more often than it *kicks* or *turns*. A CBR system that simply selected the dominant class (dash) could gain a high classification accuracy while completely ignoring the other classes.

Instead, we use the *f-measure*. We define the f-measure, $F$, for a single action, $i$, as:

$$F_i = \frac{2 * precision_i * recall_i}{precision_i + recall_i} \tag{2}$$

with

$$precision_i = \frac{c_i}{t_i} \tag{3}$$

and

$$recall_i = \frac{c_i}{n_i} \tag{4}$$

In the above equations, $c_i$ is the number of times the action was correctly chosen, $t_i$ is the total number of times the action was chosen and $n_i$ is the number of times the action should have been chosen. The global f-measure, combining the f-measures for all $A$ actions, is:

$$F_{global} = \frac{1}{A}\sum_{i=1}^{A} F_i. \tag{5}$$

## 3 Case Representation and Comparison

As we focus on spatially-aware agents, an important issue is that of representing the agent's environment. Assuming a previous step in the considered system extracts symbolic information from a robot's sensor data, the “raw” information is a list of recognized objects with spatial coordinates, for example in polar coordinates with respect to the agent, as in the RoboCup context.

The standard inputs of most machine learning systems are *feature vectors*, and in most CBR publications (particularly in the RoboCup context) researchers have manually defined vectors from their input data, selecting features according to their expertise in the application domain. For soccer simulation, the feature vectors comprised such heterogeneous features as the distance from the ball to the net, the current score, counts of players in particular zones, etc. In previous work on this project [3,4,5] we have adopted an orthogonal approach, exploring techniques to manipulate the raw data received from the feature extraction step, minimizing application-specific bias and human intervention.

### 3.1 Raw Data Representation

The first representation that we consider here is a simple list representation of the visible objects with their exact coordinates. The main problem with this representation is that is does not form an ordered set of features of a fixed length. At a given time the agent only has a partial view of the world, and cannot necessarily differentiate objects of a given symbolic category. For example, a mobile robot navigating a city could label the objects surrounding it as cars, people, or buildings, but probably not identify each one according to some complete reference of all possible cars, people, or buildings.

Comparing two cases represented by such "bags of objects" involves comparing the sets of objects present in the scenes, matching as many objects from one set to objects of the other set, and secondly evaluating the actual physical distance in between matched objects. As permutations need to be considered, the cost of object-matching algorithms is very high, as reported by Lam *et al.* [4] and Karol *et al.*[6] in separate work. Objects of one scene which cannot be matched to an object of the other must also be accounted for, for example by a penalty added to the distance value.

### 3.2 Histogram Representation

In [5] we presented an alternative approach that creates a vector using all the spatial data without the bias of manual feature selection. Our approach takes inspiration from *grid occupancy maps* [7] used in mobile robotics, a technique where sections of the environment are assigned probabilistic indications that they contain obstacles. Such a representation aims to project all the available information (e.g. from a sonar) on a feature map that represents the entire known environment. Our representation is based on histograms of objects over a partition of the visible space, and transforms a list of objects into an image-like representation with customizable granularity.

As a feature vector, this representation supports practical similarity metrics and opens the door for the application of other machine learning techniques while avoiding the need of *a priori* manual feature selection. Distance and similarity metrics that we have experimented with include Euclidean distance and a similarity metric based on the Jaccard Coefficient, which has proved to give better results in our experiments (see [5]).

### 3.3 Fuzzy Histograms

Discretizing the visible space into intervals, although efficient in our practical experiments, has some drawbacks. Objects that are near the boundaries can be artificially separated into two different sections, whereas two relatively distant objects, at opposite ends of a section, would be lumped together. In fact, since we are removing the information of the exact coordinates of the objects, some cases which were only slightly different can now have the exact same representation. If these cases were associated to the same *solution* then it can be a way of removing redundant cases, but if the cases were associated to different solutions then the indistinguishability of the two cases becomes a problem.

In order to address these problems we can introduce fuzzy logic to the discretization. Fuzzy logic allows for the smooth spreading of the count of objects over neighbouring segments according to the actual position of the objects, and thus limits boundary effects. For fuzzy histograms, we can use the same distance or similarity metrics as for *crisp* (non-fuzzy) histograms.

### 3.4 Empirical Comparison

The case representation schemes we have described store the case features and calculate distance between cases in different manners. As such, we can expect the execution time of a CBR system to be different depending on which representation scheme we use. The goal of these experiments is to find out how large the case base can be for each representation (while still meeting our imposed 50ms time limit) and how well the CBR system performs when using a case base of that size.

The experiments will use the following parameters:

– The player we will attempt to imitate is Krislet [8]. Krislet uses simple decision-tree logic to express its behaviour, namely that the agent will always search for the ball, attempt to run towards it, and then attempt to kick it toward the goal. Although it may seem that simply inducing decision-trees from our data would be an obvious solution to imitate this agent, our preliminary studies found that this required more human intervention and performed less accurately than a case-based reasoning approach.
– All features will be given an equal weighting.
– The CBR system will use a 1-nearest neighbour algorithm.
– All case representations will work on identical datasets (although they will represent the data sets differently).
– The histogram approaches will discretize the data into a 5x8 grid.
– The histogram approaches use the Jaccard Coefficient similarity measure.

Case bases of varying sizes were used in order to determine the maximum number of cases that could be used within a 50ms timeframe. We can see that the histogram approaches can utilize far more cases, nearly 5 times more, that the raw data representation (Table 1). Also, the histogram approaches achieve higher f-measure scores in all categories except for *kicking*, with the crisp approach slightly outperforming the fuzzy approach.

**Table 1.** Comparison of case representation

| | **Max. cases** | $f1_{global}$ | $f1_{kick}$ | $f1_{dash}$ | $f1_{turn}$ |
|---|---|---|---|---|---|
| **Raw** | 3012 | 0.43 | 0.17 | 0.70 | 0.41 |
| **Crisp histogram** | 14922 | 0.51 | 0.12 | 0.84 | 0.57 |
| **Fuzzy histogram** | 14922 | 0.50 | 0.12 | 0.82 | 0.56 |

## 4 Feature Selection

The comparison between cases, usually a similarity or dissimilarity measure, occurs quite often in a single CBR cycle and can represent a majority of the computational time required by the CBR system. If the comparison between cases is a function of the features contained in the cases, such that the computation time of the comparison is proportional to the number of features, then removing unnecessary or redundant features can reduce the computational time required.

*Wrapper* algorithms [9] are a type of feature selection algorithm that search for an optimal feature weighting by evaluating the weightings when using them in a target algorithm (in our case, a CBR algorithm). This is in contrast to a *filter* algorithm [10] that selects features without using the target algorithm. Wrapper algorithms are often favoured because they directly use the algorithm that will use the feature weights, although they do have a higher computational cost.

The downside of existing wrapper algorithms, when taking into account the real-time concerns, is that they select the features that will optimize the performance of a given algorithm when using a fixed-sized training sample. One should note though that performing feature selection on a fixed-sized training sample will produce an optimum feature weighting for that training sample and will not take into account that every feature removed will result in more cases that can be evaluated. For example, the removal of a feature might not improve the performance of the target algorithm using a fixed-sized training set but the performance might be increased if that feature was removed so that more cases could be added to the training set (potentially improving the diversity of the training set).

Given the total time to solve a problem case using a CBR system, $t_{tot}$, when the CBR system uses a case base of size $N$, then the average execution time cost per case, $t_{case}$, is:

$$t_{case} = \frac{t_{tot}}{N} \tag{6}$$

And if the each case is composed of $i$ types of features, then the average execution time cost per feature type, $t_{feat}$, is:

$$t_{feat} = \frac{t_{case}}{i} \tag{7}$$

It should be noted that this assumes that each type of feature has an equal execution cost. If this is not the situation, each feature type can have a different cost value. We can then apply the following algorithm to complement the use of any existing wrapper feature selection algorithm:

**Algorithm 1.** Dynamic Training Set Feature Selection

**Inputs:** WrapperAlgorithm, allCases, timeLimit, CBRAlgorithm
**Outputs:** optimum weights

```
DTSFS(WrapperAlgorithm, allCases, timeLimit, CBRAlgorithm)
    while(!WrapperAlgorithm.optimumWeightsFound()):
        weights = WrapperAlgorithm.nextWeightsToTest()
        caseCost = 0
        for(each non-zero weight in weights)
            caseCost += execution time cost of the feature being weighed
        end loop
        estimatedSize = timeLimit/caseCost
        trainingCaseBase = randomly select 'estimatedSize' cases from allCases
        CBRAlgorithm.setWeights(weights)
        CBRAlgorithm.setTrainingData(trainingCaseBase)
        performance = CBRAlgorithm.evaluatePerformance()
        WrapperAlgorithm.returnEvaluation(performance)
    end loop
    return WrapperAlgorithm.optimumWeights()
end
```

This algorithm dynamically changes the size of the training data used by a wrapper feature selection algorithm based on the estimated computational cost of the feature set that is currently being evaluated by the CBR system.

## 4.1 Experimental Results

This round of experiments looks to demonstrate the benefit of using the feature selection algorithm discussed in the previous section (Algorithm 1). For these experiments we will use a simple wrapper feature selection algorithm, a backward sequential selection (BSS) algorithm [11], as an input to Algorithm 1. We make a slight variation to this algorithm in that it does not directly evaluate weights using the CBR algorithm. Instead, it makes the current weights it wants to test available (as in the *nextWeightsToTest()* method in Algorithm 1) and waits to receive the performance of those weights (as in the *returnEvaluation(performance)* method in Algorithm 1). This wrapper algorithm requires two parameters. The first parameter is the minimum percentage a feature set must improve over the current best feature set in order to become the new best feature set. The second parameter is the number of feature sets we examine, without finding a new best feature set, before the algorithm terminates. For our experiments we will use a 0.01% minimum increase and up to 5 non-improving feature sets.

When the BSS wrapper algorithm is used to perform feature selection, with all three case representation schemes that were used in Section 3 (using the same parameters as those experiments), we find that all three schemes find the same set of features (ball and teammate) produce the largest performance improvement (Table 2). For all case representations we see noticeable increases in f-measure values by using this subset of object types instead of using all object types. Case base sizes were selected based on the experiments in Section 3 so that when using all of the features the CBR process would take approximately 50ms. We see that the feature selection algorithm found a set of features that did not contain all of the features. By removing the features that the feature selection algorithm did not select (and removing their computational cost) the CBR system will then be able to process more cases within the 50ms time limit. Does it then make sense to perform feature selection using a training case base of a fixed size if that fixed size is chosen in order to ensure the real-time constraint *when using all of the features*?

In fact, each feature set will allow for a different maximum case base size depending on the number of features that are included. If we use the same BSS wrapper algorithm as an input to Algorithm 1 we can see that it may be more beneficial to remove a feature, and as a byproduct allow for a larger case base size, then to keep the feature (Table 3). Using a fixed sized training case base, we found *ball* and *teammate* to be the features that should be included. However, using a dynamic sized training case base we find that only the *ball* should be included. The performance using a larger case base size, by removing the *teammate* feature, was larger than the performance of using a smaller case base size and including the *teammate* feature. It should be noted that the same can not be said about removing the *ball* feature and only keeping the *teammate* feature, as that actually caused a performance decrease.

## 5 Case Selection through Case-Base Clustering and Prototyping

The next method for improving case base diversity that we examine, in this section, is prototyping. Prototyping involves replacing a set of cases with a single case (a prototype case) that is representative of the entire set. In order for any type of prototyping to occur, the case base can first be divided into a number of smaller groups. Each of these clusters must contain similar cases so that the prototyping process can successfully produce a case that represents the entire cluster. Ideally, the cases within a specific cluster will be nearly identical to each other, so that the prototypical case would be highly similar to all cases in the grouping. However, if the cases in a grouping are highly dissimilar then the prototypical case will be a less precise representation of the cluster.

Many clustering algorithms work on the assumption that the distance metric, used to calculate the distance between two data points, follows the triangle inequality [12]. While we can make this assumption for the histogram distance calculation and data representation in Section 3.2, the same can not be said for

**Table 2.** Feature Selection Using Fixed Training Case Base

| | **Num. cases** | **Features** | $f1_{global}$ | $f1_{kick}$ | $f1_{dash}$ | $f1_{turn}$ |
|---|---|---|---|---|---|---|
| **Raw** | 3012 | {ball,teammate} | 0.52 | 0.30 | 0.79 | 0.47 |
| **Crisp histogram** | 14922 | {ball,teammate} | 0.57 | 0.25 | 0.87 | 0.60 |
| **Fuzzy histogram** | 14922 | {ball,teammate} | 0.55 | 0.26 | 0.82 | 0.58 |

**Table 3.** Feature Selection With Dynamic-sized Training Case Base

| | **Max. cases** | **Features** | $f1_{global}$ | $f1_{kick}$ | $f1_{dash}$ | $f1_{turn}$ |
|---|---|---|---|---|---|---|
| **Raw** | 21084 | {ball} | 0.60 | 0.42 | 0.84 | 0.55 |
| **Crisp histogram** | 104454 | {ball} | 0.61 | 0.30 | 0.90 | 0.64 |
| **Fuzzy histogram** | 104454 | {ball} | 0.60 | 0.30 | 0.88 | 0.62 |

the raw data representation and associated distance calculation (Section 3.1). This is due to the fact that each case can contain a different number of known values for features and features can be indistinguishable from each other. The way in which indistinguishable features are "matched" between cases [3] can lead to such a situation. For example, consider three cases *A, B* and *C* which contain features *a, b* and *c* respectively. We might find a situation where *a* and *b* are matched when comparing *A* and *B*, *a* and *c* are matched when comparing *A* and *C* but *b* and *c* are *not* matched when comparing *B* and *C*. This situation can lead to the triangle inequality not holding true. One type of clustering algorithm that could work on such data would be non-parametric clustering algorithms [13,14].

For the histogram representations we use a k-means clustering algorithm [15] to cluster the data. However, due to the distance calculation used by the raw data representation the k-means algorithm (along with a substantial number of other clustering algorithms) can not directly be applied due to the fact that each case contains a different number of known values for features. The data must first be transformed by converting it to a distance vector [13]. The distance vector for a case contains the distance between that case and each case in the case base. So if the case base contains *N* cases, then each case will be represented by a distance vector of size *N*. After this transformation is performed we can then apply the k-means algorithm to the distance vectors.

Assuming we can adequately cluster our case data, we will now examine two possible methods to create prototypical cases from the clustered data.

## 5.1 Using a Cluster Member

The simplest method for creating a prototypical case from a cluster of cases is to simply use a single case from the cluster, a cluster member, as the prototypical case and discard the remaining members of the cluster. This method is useful because it does not require creating a new case, but instead it reuses an existing case. By avoiding the creation of a new case the case base is guaranteed to be composed entirely of acquired cases.

For a cluster with $n$ cases in it and containing the cases $\{C_i, \ldots, C_n\}$, we locate the prototypical case as:

$$C_{prot} = \arg \min_{C_i=C_1}^{C_n} \sum_{j=1}^{n} d(C_i, C_j) \tag{8}$$

This will find the case that is the minimum distance (where the distance between two cases is $d(C_i, C_j)$) from all other cases in the cluster. Likewise, we could modify the equation to find the maximum value when calculation a similarity between the cases.

## 5.2 Creating an Average Case

The second method of creating a prototypical case from a cluster of cases is to create an "average case". This entails determining an average position that spatial objects will be located when examining all cases in the cluster. Compared to the first method, this method constructs a novel case and does not reuse an existing case. The process of creating an average case when all cases have a fixed number of features is quite simple (the average case will contain the average value of each feature). The averaging process becomes more difficult in situations where cases can have different numbers of features. For example, with the raw data representation (Section 3.1) the issue of matching features between a pair of cases becomes substantially more difficult when matching between a set (cluster) of cases. To deal with the differing numbers of features we propose Algorithm 2.

This algorithm uses a case in the cluster that is central to the other cases (using Equation 8) and performs a pair-wise matching between that case and each of the other cases in the cluster. The resulting prototype will have the same number of features as the cental case and the feature values will be highly dependant on how the other cases matched to the central case. This means changing the case used as the cental case can result in different prototype cases being produced.

## 5.3 Experimental Results

In this section of experiments we demonstrate the results of applying the prototyping methods that we have described previously. For these experiments we will use the same parameters that we used in Section 3 and we will use the results from that section (the maximum-sized case base that can be searched in 50ms) as the benchmark.

Initially, each of the case bases (one for each case representation scheme) must be clustered. As was mention previously in this section, we will use the k-means clustering algorithm. The k-means algorithm requires a parameter, the *k-value*, to specify how many clusters the data will be partitioned into. For

**Algorithm 2.** Spatial Cluster-Average Prototype

**Inputs:** cluster of cases
**Outputs:** prototypical case

```
SCAP(cluster)
    remove the case, C, that is closest to all other cases
    for(each feature f in C):
       create an empty list and add the feature to that list
    end loop
    while(more cases exist in cluster):
       remove the next case in the cluster, N
       for(each feature in C):
          match the feature with a feature in N
          add the feature value from N to the list for this feature
        end loop
    end loop
    create a prototypical case, P, that contains no objects
    for(each feature in C):
       compute the average location of all features in the feature list
       add a feature with the average location to P
    end loop
    return P
end
```

each representation this value was found experimentally by finding the smallest k-value such that the resulting clusters were still homogeneous. A cluster is homogeneous if each case in the cluster has the same solution (the same action in the RoboCup domain).

With the resulting clusters we then applied the prototyping methods described in Section 5.1 and Section 5.2. Both prototyping methods resulted in the same decrease in number of cases and decreased execution time, since they used the same data partitioned into the same number of clusters. The difference between prototyping using a cluster member and prototyping using an average case (Table 4) can be seen in their influence on performance. While both methods result in a slight decrease in the f-measure, a tradeoff for the decrease in execution time, we see that the f-measure decreases less when prototyping using an average cluster.

Although we have focused exclusively on prototyping a case base that can already be searched within our real-time limit of 50ms, in order to compare with benchmarks set in previous experiments, the real benefit of such an approach is to compress a larger case base to fit within the time limit. Given that k-means allows the number of clusters produced to be specified, the case base can be partitioned into a number of clusters equal to the maximum allowable case base size. Each cluster can then be used to create a prototype case and the resulting case base will be exactly the maximum allowable size. The only limiting factor is that the more a case base is compressed the larger the potential performance decrease that will occur.

**Table 4.** Prototyping using cluster member and average case

| | **Initial cases** | **Final cases** | **Initial execution time** | **Final execution time** | **Initial** $f1_{global}$ | **Member Final** $f1_{global}$ | **Ave. Final** $f1_{global}$ |
|---|---|---|---|---|---|---|---|
| **Raw** | 3012 | 2642 | 50ms | 44ms | 0.43 | 0.41 | 0.42 |
| **Crisp** | 14922 | 12335 | 50ms | 41ms | 0.51 | 0.47 | 0.49 |
| **Fuzzy** | 14922 | 11642 | 50ms | 39ms | 0.50 | 0.47 | 0.49 |

## 6 Related Work

In our previous work [4,3,5] we have exclusively looked at the application of CBR to the topic of agent imitation and the various algorithms and data representations used to facilitate that process. In that work we made no attempt at preprocessing the case base in order to improve search time and case base diversity.

Using CBR in the domain of RoboCup has been explored numerous times before, including the simulation [16,17,18], small-sized robot [19] and four-legged robot [20,6] leagues. These CBR systems are often given a complete world-view [17,18,19,6] or an artificially increased world-view (from opponents that communicate their location [20]) which may not be attainable in many real-world domains. The features contained in a case are selected by a human expert [16,17,18,19,20,6] and the cases are often authored [19,20] or filtered [16] by a human expert. The main difference between our approach and these works is that we use automatically generated cases, by observing another agent, that contain all objects the agent can see, thus reducing bias by avoiding feature selection by a human expert.

The topic of feature selection and feature weighting has been covered extensively in case based reasoning research. This work ranges from comparisons of various feature weighting methods [21] to applications of feature weighting [22]. While substantial work exists in the field of CBR, to our knowledge ours is the first to address the feature selection needs of a real-time agent. Similarly, prototyping and generalization have previously been examined and applied in CBR [23] but the way in which a spatial-aware agent represents a case complicates the prototyping processes and requires special consideration. Lastly, while it may appear that we ignored topics related to case base structure [24] and fast-indexing techniques, we feel that our techniques compliment such work and should be used in combination with, rather than as an alternative to them.

## 7 Conclusions

Throughout our case study we have experimented using three different methods of case representation. Our results have generally shown the histogram representations to significantly outperform the raw representation (using a t-test with p=0.01) and the crisp histogram outperformed the fuzzy version (p=0.05). There are two notable exceptions. Throughout we have found that the raw representation achieves higher f-measure values related to the action of kicking (p=0.01). This is likely because the artificial cell boundaries created by the histogram approach. Secondly, we find that the fuzzy histogram representation achieves better reduction in size while having no significant difference in f-measure score compared to the crisp histogram. This is because the fuzzy histogram approach produces fewer cases that have the same representation but different actions, since the fuzziness better encodes the exact position of features.

We have demonstrated throughout this paper that applying preprocessing techniques to a case base can increase the performance of a CBR system by allowing it to consider more cases within a real-time limit or increasing the diversity of the case base. We examined modifying the distance calculation used (as a byproduct of changing the case representation), selecting a feature set that optimize performance and clustering similar cases so that they may be used to generate a prototypical case. While we have shown these techniques to be successful in the domain of RoboCup soccer, and more specifically the imitation of soccer playing agents, they are applicable to any situation where a CBR system uses spatial data with real-time concerns.

Although the f-measure values attained may seem low (0.5 - 0.6 range), successful agent imitation can be observed when watching the imitative agent play a game of soccer. This study focused on data from a single agent (due to space limitations) but further analysis for agents of different complexity, data sets, source code and game videos can be found at http://rcscene.sf.net.

## References

1. Smyth, B.: Case-base maintenance. In: Proceedings of the Eleventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (1998)
2. RoboCup: Robocup online (2008), `http://www.robocup.org`
3. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based approach to imitating robocup players. In: Twenty-First International FLAIRS Conference, pp. 251–256 (2008)
4. Lam, K., Esfandiari, B., Tudino, D.: A scene-based imitation framework for robocup clients. In: Proceedings of the Workshop MOO at AAMAS 2006 (2006)
5. Davoust, A., Floyd, M.W., Esfandiari, B.: Use of fuzzy histograms to model the spatial distribution of objects in case-based reasoning. In: Bergler, S. (ed.) Canadian Conference on AI, pp. 72–83. Springer, Heidelberg (2008)
6. Karol, A., Nebel, B., Stanton, C., Williams, M.A.: Case based game play in the robocup four-legged league part i the theoretical model. In: RoboCup (2003)
7. Moravec, H., Elfes, A.E.: High resolution maps from wide angle sonar. In: Proceedings of the 1985 IEEE International Conference on Robotics and Automation, pp. 116–121 (1985)
8. Langner, K.: The Krislet Java Client (1999), `http://www.ida.liu.se/frehe/RoboCup/Libs`
9. Kohavi, R., John, G.H.: Wrappers for feature subset selection. Artificial Intelligence 97(1-2), 273–324 (1997)
10. John, G.H., Kohavi, R., Pfleger, K.: Irrelevant features and the subset selection problem. In: Proceedings of the Eleventh ICML, pp. 121–129 (1994)

11. Aha, D.W., Bankert, R.L.: A comparative evaluation of sequential feature selection algorithms. Learning from Data: AI and Statistics V, 199–206 (1996)
12. Xu, R., Wunsch, D.I.I.: Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3), 645–678 (2005)
13. Bicego, M., Murino, V., Figueiredo, M.A.T.: Similarity-based clustering of sequences using hidden markov models. In: Perner, P., Rosenfeld, A. (eds.) MLDM 2003. LNCS, vol. 2734, pp. 86–95. Springer, Heidelberg (2003)
14. Dubnov, S., El-Yaniv, R., Gdalyahu, Y., Schneidman, E., Tishby, N., Yona, G.: A new nonparametric pairwise clustering algorithm based on iterative estimation of distance profiles. Mach. Learn. 47(1), 35–61 (2002)
15. Hartigan, J.A.: Clustering Algorithms. John Wiley & Sons, Inc., New York (1975)
16. Berger, R., Lämmel, G.: Exploiting past experience – case-based decision support for soccer agents. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667. Springer, Heidelberg (2007)
17. Steffens, T.: Adapting similarity measures to agent types in opponent modeling. In: Proceedings of the Workshop MOO at AAMAS 2004, pp. 125–128 (2004)
18. Ahmadi, M., Lamjiri, A.K., Nevisi, M.M., Habibi, J., Badie, K.: Using a two-layered case-based reasoning for prediction in soccer coach. In: Proceedings of the MLMTA 2003, Las Vegas, Nevada, pp. 181–185 (2003)
19. Marling, C., Tomko, M., Gillen, M., Alexander, D., Chelberg, D.: Case-based reasoning for planning and world modeling in the robocup small sized league. In: IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments (2003)
20. Ros, R., de Mántaras, R.L., Arcos, J.L., Veloso, M.: Team playing behavior in robot soccer: A case-based approach. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 46–60. Springer, Heidelberg (2007)
21. Wettschereck, D., Aha, D.W.: Weighting features. In: First International CBR Research and Development Conference, pp. 347–358. Springer, Berlin (1995)
22. Jarmulak, J., Craw, S., Crowe, R.: Genetic algorithms to optimise CBR retrieval. In: 5th European Workshop on Advances in CBR, pp. 136–147 (2000)
23. Maximini, K., Maximini, R., Bergmann, R.: An investigation of generalized cases. In: Ashley, K.D., Bridge, D.G. (eds.) ICCBR 2003. LNCS, vol. 2689, pp. 261–275. Springer, Heidelberg (2003)
24. Lenz, M., Burkhard, H.D.: Case retrieval nets: Basic ideas and extensions. Kunstliche Intelligenz, 227–239 (1996)