FIRA Simurosot: A CBR approach for defensive strategies

Surajkumar Harikumar (EE11B075) and Manikandasriram S R (EE11B127)

Department of Electrical Engineering, IIT Madras

Abstract. For this project, we implement a defensive CBR agent for decision making in a 3-defender 1-attacker robosoccer setup. The CBR system uses intuitive features to decide among several high level actions to execute. We implement the CBR system based on our custom similarity measure, and we acquire a dataset of cases by running the game over multiple rule-based agents. We explore a footprint case representation for our dataset, and discuss the strengths and drawbacks of this method.

1 Introduction

The FIRA simurosot competition is a simulation league which pits 2 computer agents against each other in a 5vs5 soccer game. Teams from across the globe design strategies that compete against each other in a near real-time simulator. There are several competitions of this form, but the distinguishing factor of Simurosot is the speed at which thee game is played. The pace of the game allows for suboptimal decision choices, and so most teams implement a rule based system, reprogrammed for each robot. The issue with that is inability to adapt to new opponents and the very manual nature of decision making.

Our project is to implement a CBR based decision making agent for the simurosot league. Several of the lower level modules such as path planning, obstacle avoidance, simulation, have been implemented as part of Team Sahas, who represent IIT Madras at FIRA every year. Several higher level behaviours such as steal ball, drive to goal, clear ball have also been implemented. So the task of the CBR agent would be to make decisions on high level strategies based on the game state. CBR provides tremendous value in this context, as good actions can be learnt based on past games, and the actual implementation of these tasks will still depend on the current game state. Viewed this way, CBR performs the role of an aggregator for many rule based approaches, allowing the strategy to adapt based on the game conditions .

The specific task we focus on here is the design of a defensive agent for Robosoccer. For this, we assume a single opponent attacker, 2 defenders and a goalie, with the task of finding good actions for the defense team, given a game state. The focus of our project is to find a good feature space representation of the problem (which is relevant to the decision making), the generation of a minimal case base (using footprint case methods), and the analysis of case base competence.

The outline of the report is as follows. We first discuss the details of the CBR implementation, namely the feature space representation, similarity measures, and justification for the same. We then move on to the case base acquisition, which was done by running several games on our simulator and gathering data. We then discuss the generation of footprint cases from the large case base obtained above. The whole system was implemented in C++ in tandem with the Team Sahas simulator and base agent.

2 defCBR - Our Defensive CBR implementation

2.1 Derived Feature Space

We now discuss defCBR, our implementation of the defensive CBR agent. The problem space is inherently defined by many features, which include the positions and velocity vectors of each robot, and other details such as ball possession. It is very difficult to make decisions based solely on these properties. As humans, we consider factors like distance of players from the goal, relative speed of the ball angle. It is also easier for the CBR agent to make decisions on actions based on these parameters.

We incorporated a derived feature space which uses the following features :

- Ball position and velocity w.r.t goal
- Position of two nearest defenders and attacker w.r.t ball
- Angle of ball w.r.t bots direction
- Goalie position w.r.t ball

This set of 16 features were deemed to be the most important ones for decision making. We can intuitively make decisions based on these metrics. If the ball is very close to the goal, the defenders would blank the ball and the goalie would be on high alert. If the ball were further away, the defenders would oscillate near the goal and possibly steal the ball.

These were also chosen in such a way that they cover the entire problem space, spanned by the initial features. The only features not incorporated are the velocities of the defender. This was a design choice we made, justified by the fact that robots are differential drive, and can change velocity very quickly in a set direction. The angle which the robots point towards is important since turning is a more difficult task.

2.2 Solution Space

The solution space for our CBR system is actions and behaviours. Loosely put, an action is an abstract behaviour which triggers a series of events in the system. These are usually made to be human interpretable. Some example actions:

- StealBall. Executing this action would involve locating the ball, the nearest opponent, setting a target point behind the ball, and implementing a path planner with obstacle avoidance to reach there. It then turns to face the ball.

– DriveToGoal - This involves going behind the ball, turning, facing the ball, approaching it, and pushing towards the goal, while avoidnig other players and the goalie

There are many steps that go into each of these actions, all of which have been implemented as part of the Sahas Project. So it is sufficient for the CBR system to give high level strategy decisions in terms of the abstract notion of an action.

Some actions used are:

– goalie-block-ypos - Track and block the y position of the ball
– goalie-block-expected-ypos - Based on the ball's trajectory estimate where it would reach the goal, and block that point. Particularly useful for fast moving ball case.
– goalie-clear-ball - If the goalie is in possession, clear the ball
– defender-steal-ball
– defender-clear-ball
– defender-spin-clear
– defender-block-line
– defender-sine

There are 2 kinds of actions:

– Non-parameterized - These actions are implemented solely based on the game state. Example: goalie-block-ypos
– Parameterized - These actions have some parameters, which determine how it is performed. Example: goalie-clear-ball requires the exact angle and distance to clear the ball

Our CBR system seeks to choose which among these actions each player should perform. It is quite a nuanced problem, say whether and how to clear the ball, and when to spin clear. Also, for a parameterized action, the CBR system provides a solution which might not be the best solution for the problem. In this case, adaptation knowledge is required to modify the retrieved solution.

2.3 Similarities

Local Similarities For each vector, we calculated a distance function, Euclidian distance for the position vectors, and ratio metrics for the velocities. For the distance fuction thus computed, we used an **approximate sigmoid function** given by

$$f(d) = \frac{1}{2}\left(1 - \frac{k}{1+|k|}\right); \quad k = \frac{d-\theta}{\alpha} \tag{1}$$

where θ is the point of similarity 0.5, and α gives the rate of decay of the sigmoid. This was chosen because of its closeness to the sigmoid, and because *abs* is faster than *exp*. Figure (1) shows a plot of this function.

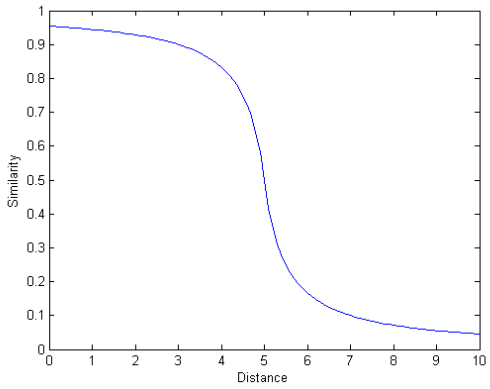

Fig. 1: Approximate Sigmoid used for Local Similarity Measure $\theta = 5, \alpha = 0.5$

Global Similarities For net similarity , we assume a weighted linear combination of local similarities. We also assume these weights are static numbers, independent of the features. To do this, we first generate a preference order on the feature importance, and assign weights based on this. We pre normalize the data to account for scale difference.

The preference order and weights are given as:

- Ball Distance and Ball velocity - 110
- Distance from nearest attacker to ball, angle between attacker and ball - 75
- Distance from nearest defender to ball - 50
- Distance from second defender to ball - 40
- Angle between nearest defender to ball - 30
- Distance from goalie to ball - 20
- Angle between second defender to ball - 50

we see that in general distances matter more than angles, and that ball position matters most. The choice of θ, α for each is a decision variable. We divided our 2200×1800 field into 50×50 grids, so that points within the same grid have are more than 0.5 similar. SO theta distance was chosen as $50\sqrt{2}$.

2.4 Retrieval

The retrieval was implemented by computing the similarity for all cases in the case base for a given problem, and generating a priority queue. The top k entries in the queue are the most similar k cases.

Since there are only a small number of actions, and because of the continuity in movement, we know that several (especially consecutive) cases will have the same solution. So a smaller representation of the case base would be sufficient.

2.5 Adaptation

One important point regarding our system is adaptation. The output given by the system is an action, which is executed based on the problem game state, not the retrieved case's. So there is some inbuilt adaptation for the problem, due to the abstract nature of output. There are parameterized cases, whose action also partially depends on the case data. In this case, we retrieve multiple solutions, and check for the validity of the solution. One suggestion here is to use a weighted average of parameters of retrieved cases for adapted value. A solution is chosen among the closest k retrieved solutions based on time to execute.

3 Footprint Analysis

After we acquired a set of about 1000 cases, our next task is to find a small set of cases that represent the case space. For this, we use the idea of Footprint analysis [2].

$$\begin{aligned}
\text{RetrievalSpace(t)} &= \{c \in C : \text{c is retrieved for t }\} \\
\text{AdaptationSpace(t)} &= \{c \in C : \text{ c can be adapted for t}\} \\
\text{Solves(c,t) iff } &\{c \in \mathit{RetrievalSpace}(t) \cap \mathit{AdaptationSpace}(t)\} \\
\text{CoverageSet(c)} &= \{c' \in C : \text{Solves(c,c')}\} \\
\text{ReachabilitySet(c)} &= \{c' \in C : \text{Solves(c',c)}\}
\end{aligned} \tag{2}$$

The Relative Coverage is then defined as

$$\text{RelativeCoverage(c)} = \sum_{c' \in \text{CoverageSet(c)}} \text{ReachabilitySet(c')} \tag{3}$$

So if we have the notion of RetrivalSpace and AdaptationSpace, the relative coverage can be computed.

- RetrievalSpace - c is said to be retrieved for t if it is in the top k cases retrieved for the problem t
- AdaptationSpace - The notion of adaptation space is a little more difficult, since there is little need adaptation knowledge in our system. So we say a case can be adapted to solve a problem if its solution doesn't completely solve the problem.
- This is meant for weeding out the false positives, such as a nearby case retrieved indicating the ball can be cleared, while in reality there is an attacker along that very path.

To compute the above quantities, we generate a **directed hypergraph** based on Solves(). Every case represents a node, and the edges represent Solves(). Using

this, the Coverage and Reachability sets are given by the in-edges and out-edges from a given node.

Relative Coverage in this sense becomes a PageRank like metric. We can use this to compute the importance of each case to the case base. We can then set a threshold and only include cases relative coverage larger than a certain number, while maintaining problem space coverage.

- RelatedSet(c) = CoverageSet(c) $\cup$ ReachabilitySet(c)

- SharedCoverage(c1,c2) iff RelatedSet(c1) $\cap$ RelatedSet(c2)

We can then define a Competence Group (maximal sets of independent cases with shared coverage). In every competence group, we can find the cases with highest relative coverage that solve all cases in group. This gives the footprint set.

4 Implementation

4.1 Data Collection

We used several strategies, ran multiple instances of the soccer game, and sampled every 0.5 seconds to gather cases. A sample strategy used for collecting data from the system

- **Goalie**
  - If *OurBall* and *goalie_is_closer*, then *goalie_clear_ball* else *goalie_defend_ball_ypos*
  - if *not OurBall* and *on_goal*, then *goalie_defend_ball_extrapolated_position*, else *goalie_defend_ball_ypos*
- **Defenders**
  - If *OurSide* and *not OurBall*, then *defender_steal* else *defender_pass_clear*
  - If *not OurSide*, then *defender_sine*
- **Attacker**
  - If *TheirSide*, then *move_ball_to_goal*

In total, we gathered a total of 663 cases, with the 15 dimensional feature space. We then used our CBR system to find similar cases for a given problem. We also set a manual threshold of $globalsim > 0.7$ for similar cases.

Figure (2) shows some of the retrived cases for a given problem. We see that the cases retrived here are quite close, indicating a reasonable similarity measure. We expermiented with some values of this threshold before choosing 0.7, and set $k = 6$ for the rest of our experiment.

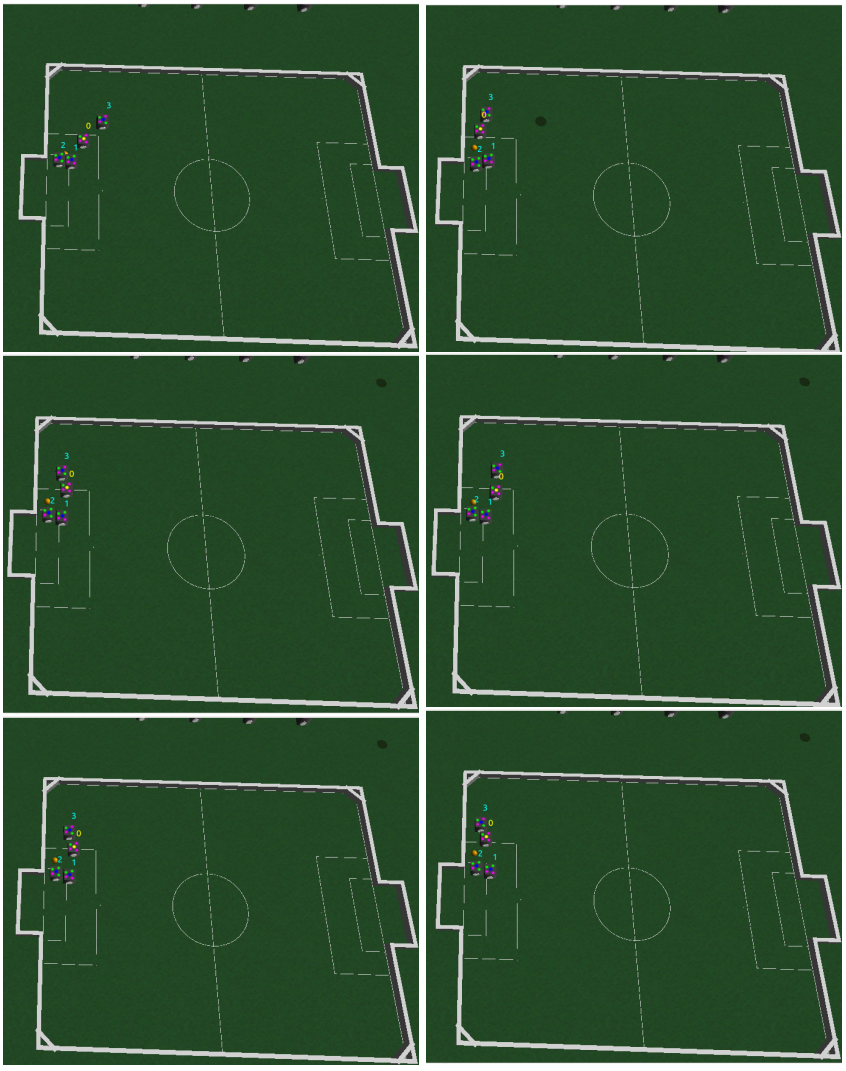

Fig. 2: Retrieved cases for a problem

Two important aspects to training the system:

- Random restarts from different initial configurations
- Running different strategies for attacker and defender
- If the different strategies produce contradicting behaviours, the conflicts need to be resolved

We ran our case base generator using a very small number of training strategies. We require more strategies to ensure proper coverage of the problem space. One related problem with this is that 2 different strategies could have different actions for a very similar problem.

One way to resolve this is to obtain a large data set using multiple strategies, cluster the data, and manually resolve some of the conflicts. We could also use this to make our data set little smaller, and this preprocessing is required for the CBR system and footprint computation. The whole premise of CBR is 'similar problems have similar solutions', and this help ensure that. A more consistent case base would also have a smaller number of footprint cases.

5 Inferences

We then set to the task of footprint based retrival. For each case, we ran the retrival, found the closest 6 cases, and formed a directed graph using these. An image metaphor for the graph is shown in Figure (3). This is a visualization of a 663×663 grid, where a white dot represents that a case on the row solves the case on the column.

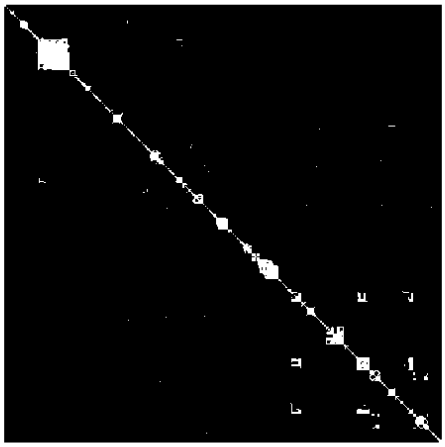

Fig. 3: Related Set as Image

There are some interesting inferences we can make from this metaphor. To a large extent, it is symmetric which makes intuitive sense since our notions of similarity are symmetric. Also, cases seem to form rectangles in the metaphor. This also makes sense, since we acquired data during runs of a game, and the game state doesn't change very quickly. There are square corner like regions, indicating that a similar scenario occurred later.

We can use this to compute the case alignment metric [3]

The graph seems very sparse at this point. This is because we have a very rigid definition of RelatedSet. Since $k = 6$ every row or column has degree atmost 6. By relaxing this assumption, and trying multiple strategies, the graph will get filled out more.

We then computed the relative coverage metric for each case. this reflects the importance of a case. based on the number of cases it solves, and their importance. Figure (4) shows the relative coverage for the case base. The peak relative coverage is about 21, and this seems to coincide with the square region at the start of the case image.

We then computed the sharedCoverage set for each of these. The computation of the competence group is a little more difficult. The framework perform these tasks is in place to. Once done, this could generate footprint cases.

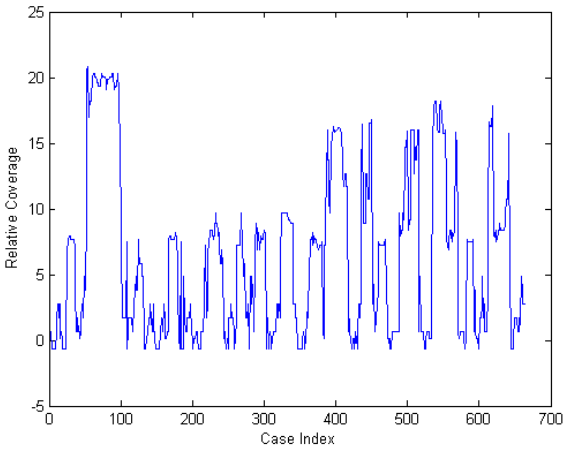

Fig. 4: Relative Coverage

5.1 Limitations

One point to note here is that the number of footprint cases is atleast equal to the number of competence groups, which for our data set appears to be large. For a better data set, with higher k, and more strategies, the number of competence groups should reduce. We want the number of competence groups to be as small as possible. Also, we have a very small number of actions in our set, so we must also consider the possibility of multiple competence groups having a similar action, in which case we must possibly modify our similarity metric.

6 Conclusion

Through this project, we implemented a defensive CBR system for FIRA robosoccer. We defined a more intuitive feature set, similarity metrics, abstract actions (such as steal, clear), inherent adaptability, and implemented the CBR retrieval engine in C++.

We used several rule based strategies to collect a case base of 663 feature-action sets. This was used to drive decision making, and with lower level execution handled by other modules, a match was played using the agent. We show how similar several retrieved cases are in the actual game.

We switched to the generation footprint cases. Based on the notion of cases solving other cases, we made a directed graph (as image metaphor), and computed the relative coverage (importance metric) for each case. Some inferences from here are the largely temporal nature of cases (similar cases occur closer together), and the sparsity of the RelatedSet matrix. This wass attributed to the small number of retrieved cases for any case.

Given a larger and more exhaustive dataset of cases, generated using multiple strategies and which cover the problem space well, we can generate a set of footprint cases. These footprint cases would take the best actions across multiple rules, and we can use this to drive decision making for a defensive robosoccer system.

References

1. Lopez De Mantaras, Ramon, et al. "Retrieval, reuse, revision and retention in case-based reasoning." The Knowledge Engineering Review 20.03 (2005): 215-240.
2. Smyt, Barry, and Elizabeth McKenna. "Footprint-based retrieval." Case-Based Reasoning Research and Development. Springer Berlin Heidelberg, 1999. 343-357.
3. Chakraborti, Sutanu, et al. "Visualizing and evaluating complexity of textual case bases." Advances in Case-Based Reasoning. Springer Berlin Heidelberg, 2008. 104-119.