A Case-based Reasoning Approach to Imitating RoboCup Players

Michael W. Floyd and **Babak Esfandiari** and **Kevin Lam**
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario

Abstract

We describe an effort to train a RoboCup soccer-playing agent playing in the Simulation League using case-based reasoning. The agent learns (builds a case base) by observing the behaviour of existing players and determining the spatial configuration of the objects the existing players pay attention to. The agent can then use the case base to determine what actions it should perform given similar spatial configurations. When observing a simple goal-driven, rule-based, stateless agent, the trained player appears to imitate the behaviour of the original and experimental results confirm the observed behaviour. The process requires little human intervention and can be used to train agents exhibiting diverse behaviour in an automated manner.

Introduction

A software agent's behaviour can be characterized by its reaction to a sequence of inputs over a period of time (Wooldridge 2002) given a configuration of surrounding entities (Murakami *et al.* 2003). Creating and inputting such behaviour can be very time-consuming, and involves a significant amount of technical knowledge (for example, programming skills) and domain expertise.

Instead, we want to make our agent learn its behaviour by observing and imitating another agent. Imitation is a useful technique when one wants to recreate existing behaviour for which the code is not available, and/or when one lacks the time or skills to do it manually. Our goal is to address the problem of training an agent in the RoboCup domain, as we believe that it is a good environment to obtain data easily and experiment with agents that require spatial and temporal knowledge representation and reasoning. Imitation in the RoboCup domain can be used to quickly create "sparring partners" with specific behaviour of interest to train against, or one could even create a new agent from scratch by putting the user "in the agent's shoes", guiding the agent during practice, recording the traces of such a run and using it as the reference behaviour to imitate.

By using a case-based reasoning approach we are able to store the agent's spatial knowledge as *cases* and apply a cyclical reasoning method (Aamodt & Plaza 1994) to determine the actions the agent will perform.

There is a known set of competent teams from the annual Robot World Cup games, and it should ideally be possible emulate existing teams with as little human intervention as possible during the training process. Our goal is not to win the competition but to evaluate the feasibility of quickly developing agents through this approach. We could then take the imitative techniques we have developed and apply them in various other domains (robotics, game development, etc.).

RoboCup

The RoboCup (Robot World Cup) Simulation League (RoboCup 2007) is a unique testbed for the design of autonomous software agents, where agents represent players on a soccer team. Typical RoboCup agents collaborate with teammates, use high-level strategies such as goaltending, defence or offensive plays, and work toward a team goal while also thwarting the opponents' goals. There is a wide body of existing research to draw from, which implies much available data for case-based reasoning approaches.

RoboCup is a real-time, dynamic and non-deterministic environment. Client agents in RoboCup 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. 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, goals, or the numerous lines and flags located on the field. Objects have attributes such as distance, direction, and identifying labels (e.g., a player's team and uniform number).

Clients then send a command to the server in response, typically to perform an action such as a `dash`, `kick`, or `turn`.

Current Approaches

Current attempts to use case-based reasoning for RoboCup agents (Ros *et al.* 2007; Wendler & Lenz 1998; Marling *et al.* 2003; Steffens 2004) have focused on using CBR for high-level team activities, like executing set plays given the configuration of players on the field. The major limitation of

Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

these approaches is that they require a complete view of the soccer field and therefore they make domain-dependent assumptions and do not have to deal with an incomplete world model. As well, they require extensive expert knowledge, for both defining the features that compose a case and the creation of cases. The most promising use of CBR for a single agent has been with the goalie agent (Berger, Hein & Burkhard 2007; Marling *et al.* 2003) although other techniques are usually used in combination with CBR (Berger, Hein & Burkhard 2007).

Existing attempts at generating an agent's behaviour from direct observation such as Matsui's ILP agent (Matsui, Inuzuka & Seki 2000) require prior processing including generation of predicates or other knowledge, and properly classified examples from which to train. Typically, agent observation is only used to match previously-generated profiles, but methods for on-line opponent modelling exist (Drucker *et al.* 2002).

Imitation learning has also been applied to imitating reactive behaviour in first person shooter style games (Bauckhage, Thurau & Sagerer 2004).

Contributions

We have developed an extensible framework for research in RoboCup agent imitation, which includes the following components:

- a RoboCup client agent that uses the case-based reasoning framework to imitate the behaviour of other agents;
- a set of algorithms that enable case-based reasoning to be performed by individual imitative agents;
- an automated process for acquiring case data, learning feature weights and performing imitative behaviour.

We provide experimental and reproducible results that demonstrate an agent's ability to imitate the high-level behaviour of an agent (and its limitations), with minimal human intervention.

The currently implemented agent is stateless, which imposes limits on the extent of which the algorithm can imitate other agent behaviours. This work addresses the challenges that must be addressed first, while future work will take on the learning of context and state-based behaviour that would further improve the effectiveness of this approach.

Methodology

Our case-based imitation framework follows a process that includes the following objectives:

1. *Perform data capture from logs generated by existing RoboCup clients.* Logs describe games as seen from *each player's* point of view.
2. *Store the captured data in a spatial knowledge representation format*, capturing "cases" — snapshots of an agent's vision and behaviour at discrete times.
3. *Preprocessing phase*, which can include feature weighting, case reduction or other offline processing of the data. In this paper we present an automated feature weighting scheme.
4. *Apply a similarity algorithm to compare real-time situations with previously-stored ones.* The new RoboCup agent responds to each situation by using the same actions that other RoboCup clients used in "similar" situations.

The resulting agent should exhibit behavioural similarities with the training agents, which are measured both qualitatively (observed similarity) and by using strict quantitative statistical measures (something that other related work often lacked).

A Case Representation For RoboCup

A *case* represents a discrete-time snapshot of a given agent's view and the associated actions taken by the agent. Each case c is then an aggregation of all visible objects (represented by their type, position, velocity and direction) and the subsequently performed action.

Conveniently, the RoboCup server provides a `see` message that describes the player's visual information. Players send commands back to the server in response. A case can then be built using the player's visual information as well as the corresponding action command they send back to the server. An entire game is thus represented as a sequence of cases $C = \{c_0, c_1, c_2, \cdots, c_n\}$.

Graphically, a case depicts the objects visible to a RoboCup player, as in Figure 1. Cases provide a convenient atomic unit through which data can potentially be further manipulated.

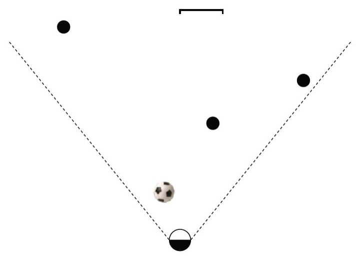

Figure 1: Visual data in a RoboCup case (dashed lines represent the field of vision).

Case Recognition

Given a new situation N and a set of observed cases $C = \{c_0, c_1, \cdots, c_n\}$, if any case $c_i \approx N$, then the recorded action from that case can be reused.

Case Recognition Algorithm

We use a k-nearest-neighbour search to find the k best case matches (in tie situations, the first encountered case is given priority); these are then evaluated using a separate action selection algorithm to decide which action will ultimately be chosen. The case recognition process follows the general algorithm shown below:

```
load the stored cases from disk
while(game on):
     get new incoming ``see'' data from server
```

```
    convert data to case object N
    for each stored case x:
        d = DistanceCalculation(N, x)
        S = keep k most similar results
    end loop
    a = ActionSelection(S)
    send action a to server
end loop
```

Note that, in RoboCup, for the action to be taken into account within a single time cycle, the agent must send an action before the end of the time cycle (30 ms). In order to ensure the agent always sends an action in time, the number of cases in the case base must therefore be below a certain threshold. The case base size threshold is determined by calculating the mean time to perform a single distance calculation (the estimated cost per case) and then computing how many cases could be examined in a single time cycle. This threshold (in our case, about 3000 cases per cycle) will limit the diversity of the case base that is used and affect the accuracy of the imitative behaviour (we have noted a 5 to 8 percentage point degradation).

The algorithm relies on two important functions, namely `DistanceCalculation` and `ActionSelection`. They are described in the following sections.

Distance Calculation

The `DistanceCalculation(N, x)` function determines a value of "similarity" between two cases. Since cases are collections of multiple objects in space, the objects are paired, and their distances are summed.

Objects on the field are all described with a distance r and a direction θ relative to the observing agent. This can be represented by a polar coordinate $P(r, \theta)$, and distances between similar objects in two distinct cases can readily be calculated.

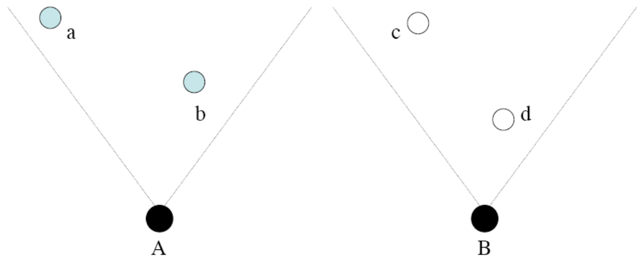

Figure 2: Two closely-matching cases.

Object Matching

The main complexity of the distance calculation algorithm actually lies in mapping pairs of objects between scenes when there are more than one object of the same type in a scene, as in Figure 2. The problem is that of minimum-weight matching in a bipartite graph, where the cost function to be minimized is the distance between each pair of objects.

In other approaches to applying CBR to RoboCup (Ros *et al.* 2007) and agent imitation (Bauckhage, Thurau & Sagerer 2004), the object matching problem was simplified by considering a fixed subset of objects (that were always visible to the agent) or by artificially ensuring that the agent possessed a complete world model. These simplifications pose two main problems. Firstly, in most domains an agent will not have a complete world model so it may not be accurate to assume such (even in RoboCup a complete world model is not available and had to be artificially created by having opponent players provide their identification and location). Secondly, using a subset of objects (that are always visible to the agent) may lead to a decrease in performance because the number of features that the agent can use for reasoning is severely limited.

We implemented two different algorithms that address object matching. The first one, known as the "blossom" algorithm (Cook & Rohe 1999) provides the optimal matching but doesn't perform fast enough given our real-time constraints. The other one is a faster heuristic (i.e. sacrificing matching optimality) that sorts objects in each scene based on their distance to the agent and greedily matches them pair-wise (Lam, Esfandiari & Tudino 2006).

Objects "missing" from one scene must also be taken into consideration. One approach is to apply a "penalty" distance for objects in one set that have no corresponding object in the other. This penalty should be inversely proportional to the distance; intuitively, this is a measure of the importance of a missing object.

Object Weighting

Finally, different types of objects may be weighted according to their perceived importance in a case. For example, to ignore a set of objects (i.e. lines on the field may be irrelevant to a given player), weights can be set to 0.

The distance calculation for n objects present in both cases may thus be expressed as:

$$\begin{aligned} d(c_1, c_2) &= \sum_n \left[w_n * d\left(obj_n, mat\left(obj_n\right)\right)\right] \qquad (1) \\ c_1 &= \{obj_1, obj_2, \cdots, obj_n\}, \\ c_2 &= \{mat\left(obj_1\right), mat\left(obj_2\right), \cdots, mat\left(obj_n\right)\} \end{aligned}$$

mat(obj) : returns the object that matches with *obj*, or *nil* if there is no matching object (in which case a penalty is applied)

Action Selection and Adaptation

The result of the k-nearest-neighbour search is a set of k cases, each a potential match. Each has an associated action (e.g., `dash, kick, turn`) and parameters (power, direction, etc). When $k > 1$, the most common action (majority vote) is used here.

Parameters must also be chosen for selected actions (e.g., which direction to kick in and at what strength). The simplest approach would be to just reuse the values originally stored in the matching case. Averages could also be taken: if there are n different instances of a `kick`, the average kick power and direction could be taken over the n results. But so far we have simply reused the action of the closest neighbour.

A possible improvement that we are currently implementing is for the agent to attempt to bind variables in the stored case together — i.e. was the {`kick, turn, dash`} aimed at a particular object?

Weight Calculation

When using a k-nearest neighbour classifier, using equal feature weights can result in poor performance when irrelevant or redundant features are present (Wettschereck & Aha 1995). Even if two agents use the same set of objects for reasoning they may place different levels of importance on each type of object. Using appropriate feature weights for an imitative agent is vitally important for the accuracy of the imitative behaviour of the agent.

We assume no prior knowledge of the agent that is being imitated, so feature weights must be determined automatically using the data that is available (the case base). Determining feature weights in an automated fashion has been successful in other case-based reasoning domains (Stahl 2005; Stahl & Gabel 2003) and we apply a similar methodology to imitation learning.

In particular, we use a genetic algorithm (Koza 1992) for the task of determining feature weights. A population of possible solutions (individuals) is maintained and at discrete time intervals (generations) the best solutions are kept for the next generation (and may be modified with genetic operators) while the remaining solutions are discarded. For our application, a possible solution is composed of the set of weights to be used by the k-nearest neighbour classifier (with one weight for each type of object).

We measure the fitness of each solution using the global f-measure, which is a function of the f-measures for each type of action. This helps to ensure that we are maximizing the ability of the agent to perform each type of action and not just actions that occur a disproportional number of times in the case base. We define the f-measure, *F*, for a single action, *i*, as:

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

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

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

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 *N* actions, as:

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

After a number of generations the system will evolve leaving only solutions that result in higher fitness, and therefore higher imitative performance.

Implementation

We developed an implementation of the case format and a RoboCup client based on the matching algorithms described previously. Krislet (Langner 1999) was used as a starting point. 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. The main benefit of using Krislet is that it is coded in Java, is easy to extend, and all its client-server communications and message parsing functions are already implemented.

To capture logs from existing games, the LogServer proxy utility (Marlow 2004) is inserted into the communications path between existing RoboCup agents and the server. We have verified experimentally that the use of LogServer does not introduce a "probe effect" in the results. The captured log files can then be directly converted into a case base.

Experimental Results

Table 1: Accuracy and Recall Results for Basic Agents

	Accuracy	**Recall** *dash/turn*
LineRun	93.51%	0.96/0.72
SquareRun	87.36%	0.94/0.46
ChaseBall	83.22%	0.84/0.58

Our experiments aim to determine to what extent existing agents can be imitated and what characteristics of such agents are particularly problematic. We also want to determine the impact of our automatic weight calculation scheme (as described previously) on the performance of the imitation agent.

Imitation Tests

In initial experiments, we examined agents that performed basic behaviours:

- *LineRun*: Runs from one end of the field to the other end, turns around, and runs back.
- *SquareRun*: Runs in a square pattern.
- *ChaseBall*: Follows the ball around the field.

These agents were run with equal weighting for all objects on the field and performed quite well without attempting any sort of optimization. A summary of the results can be seen in Table 1 (note that there are no $recall_{kick}$ values because these agents never attempt to kick).

For further experiments, we selected three different RoboCup agents of varying complexity:

- *Krislet*
- *NewKrislet* (Huang & Swenton 2003), which uses a state machine model to express goals; we consider a specific team implementation called the AntiSimpleton, which uses a simple attacker/defender strategy. The Attacker waits near the midfield line until the defenders pass the

Table 2: Average Weights Calculated by Genetic Algorithm

	Ball	**Goal**	**Flag**	**Line**	**Team**	**Opponent**	**Unknown**
Krislet	0.40	0.18	0.00	0.05	0.37	0.00	0.00
NewKrislet	0.38	0.10	0.00	0.22	0.30	0.00	0.00
CMUnited	0.19	0.03	0.39	0.28	0.01	0.08	0.02

Table 3: Accuracy and Recall Results for Soccer Agents

	Accuracy (Calculated)	**Accuracy (Equal)**	**Accuracy (Expert)**	**Recall** *kick/dash/turn* **(Calculated)**	**Recall** *kick/dash/turn* **(Equal)**	**Recall** *kick/dash/turn* **(Expert)**
Krislet	72.83%	68.37%	70.27%	0.51/0.84/0.52	0.11/0.80/0.32	0.16/0.82/0.39
NewKrislet	71.13%	65.70%	69.07%	0.46/0.83/0.53	0.23/0.80/0.43	0.43/0.80/0.46
CMUnited	53.93%	41.90%	51.23%	0.28/0.40/0.63	0.06/0.23/0.41	0.16/0.38/0.41

ball toward it, then attempts to score a goal (much like Krislet) before returning to its home position to await more passes from Defenders.

- *CMUnited* (Stone, Veloso & Riley 1999), a RoboCup Champion team which uses a layered learning architecture and a number of strategies including formation strategies. This team is chosen not because of any realistic hope of properly emulating the team but to determine how far we are from such a goal.

Automation Tests

In the experiments that follow, we will evaluate the performance of the imitating agent use three different weighting schemes:

1. Equal weights for all features/objects
2. Weights determined by a human expert
3. Weights determined automatically (see section on weight calculation)

Data was obtained by observing the agents play several complete games (with five players per team). From this, 350 cases were randomly selected for the case base and 3000 cases were randomly selected for testing. A 1-nearest neighbour search was used to select the most appropriate case and the action associated with that case was used without modification.

The genetic algorithm used a population of 50 solutions and ran for 1000 generations. Selection of solutions to keep for the next generation was performed using a tournament selection method (Koza 1992). Standard mutation and crossover genetic operators are used to evolve the solutions. The fitness, the f-measure, of a solution was determined by using that solution (a set of weights) as the weights used by the case-based reasoning algorithm on a set of test cases.

The average weights calculated by the genetic algorithm can be found in Table 2. The average accuracy and recall (listed as $recall_{kick}$ / $recall_{dash}$ / $recall_{turn}$) using the calculated weights, equal weights and expert weights can be found in Table 3 (the precision and f-measure statistics were also generated but have been omitted due to space limitations).

We can see that for all three agents the metrics statistics are noticeably higher using the calculated weights (compared to both the equal weights and expert weights). The mean values were analyzed using a single-sample t-test and the improvements were found to be statistically significant compared to the values for both the expert and equal weights (p-values <0.05).

Although we were already quite confident that using equal weights would not produce the optimum results, it is interesting to note that the expert weights (proposed by the authors of the agent or someone with knowledge of the inner workings of the agent) did not produce optimum results either. Often hidden factors play an important role in the imitative process. For example, in the Krislet agent there is a noticeable weight applied to the teammates. We know that Krislet does not pay any attention to these objects, but when performing imitation they prove to be important. In the case of the teammates, they are important because teammates are usually dashing in the direction of the ball and taking them into consideration helps the imitative agent track the ball location. Similar observations can be made for the NewKrislet agent weights as well.

Limitations and Future Work

The current case recognition framework is subject to the following limitations:

- Does not consider some visual information such as object velocity and direction or non-visual information such as body state, game state, or memory of previous conditions.
- Unable to utilize inter-agent communication.

Future work will address a number of the above limitations, including the following:

- Detection of "stateful behaviour" by examining the observed data and looking for cases with the same inputs yet different outputs, i.e. $\exists x_1, x_2 | C(x_1) = C(x_2)$ but $a(x_1) \neq a(x_2)$. Large numbers of such cases would suggest different states or other hidden factors influencing the outputs.

- Looking beyond individual cases, toward finding patterns or trends over sequences of cases —the agent should also be able to trigger high-level actions that occur over a sequence of time.

Conclusions

The case-based imitation methodology described here represents initial steps toward an automated process for observation and imitation of other agents.

Our results suggest that it is possible to learn the behaviour of a RoboCup player if its behaviour can be captured in a simple logical construct such as a decision tree. As well, machine learning techniques can be applied to the case-base in order to learn the importance of different objects and increase the overall imitative accuracy. This allows such simple agents to be imitated with minimal human interaction.

Although the players imitated are stateless, single-layered agents, they are fairly representative of at least some of the behaviours of more complex agents. These results are encouraging and suggest that with further development it may be possible to further increase the accuracy of imitative agents and allow them to imitate more complex behaviour. Source code of the described algorithms, data sets, complete results and videos showing our imitating agents in action can be found at http://rcscene.sourceforge.net .

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

Bauckhage, C.; Thurau, C.; and Sagerer, G. 2004. Learning Human-Like Movement Behavior for Computer Games. In *Proceedings of the Eighth International Conference on the Simulation of Adaptive Behavior (SAB04).*

Berger, R.; Hein, D.; and Burkhard, H.-D. 2007. AT Humboldt & AT Humboldt 3D Team description 2006. In *RoboCup 2006*. Springer.

Cook, W., and Rohe, A. 1999. Computing minimum-weight perfect matchings. *INFORMS Journal on Computing* 11:138–148.

Drucker, C.; Hubner, S.; Visser, U.; and Weland, H.-G. 2002. "As time goes by" - Using time series based decision tree induction to analyze the behaviour of opponent players. In *RoboCup 2001*, 325–330. London, UK: Springer-Verlag.

Huang, T., and Swenton, F. 2003. Teaching undergraduate software design in a liberal arts environment using robocup. In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*, 114–118. New York, NY, USA: ACM Press.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA, USA: MIT Press.

Lam, K.; Esfandiari, B.; and Tudino, D. 2006. A scene-based imitation framework for RoboCup clients. In *Proceedings of the Workshop "Modelling Other Agents from Observations" at AAAI 2006*, 24–31. Technical Report WS-06-13.

Langner, K. 1999. The Krislet Java Client. http://www.ida.liu.se/ frehe/RoboCup/Libs/libsv5xx.html.

Marling, C.; Tomko, M.; Gillen, M.; Alexander, D.; and Chelberg, D. 2003. 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*.

Marlow, P. 2004. A process and tool-set for the development of an interface agent for use in the robocup environment. Master's thesis, Carleton University.

Matsui, T.; Inuzuka, N.; and Seki, H. 2000. A proposal for inductive learning agent using first-order logic. In Cussens, J., and Frisch, A., eds., *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, 180–193.

Murakami, Y.; Ishida, T.; Kawasoe, T.; and Hishiyama, R. 2003. Scenario description for multi-agent simulation. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, 369–376. ACM Press.

RoboCup. 2007. Robocup online. http://www.robocup.org.

Ros, R.; de Mantaras, R. L.; Arcos, J. L.; and Veloso, M. 2007. Team playing behavior in robot soccer: A case-based approach. In *7th International Conference on Case-Based Reasoning*, 46–60. Springer.

Stahl, A., and Gabel, T. 2003. Using evolution programs to learn local similarity measures. In Ashley, K. D., and Bridge, D. G., eds., *ICCBR*, volume 2689 of *Lecture Notes in Computer Science*, 537–551. Springer.

Stahl, A. 2005. Learning similarity measures: A formal view based on a generalized CBR model. In Muñoz-Avila, H., and Ricci, F., eds., *ICCBR*, volume 3620 of *Lecture Notes in Computer Science*, 507–521. Springer.

Steffens, T. 2004. Adapting similarity measures to agent types in opponent modeling. In *Proceedings of the Workshop "Modelling Other Agents from Observations" at AAMAS 2004*, 125–128.

Stone, P.; Veloso, M.; and Riley, P. 1999. The CMUnited-98 champion simulator team. In *RoboCup 1998*. Springer-Verlag.

Wendler, J., and Lenz, M. 1998. CBR for dynamic situation assessment in an agent-oriented setting. In *Proceedings of AAAI-98 Workshop on CaseBased Reasoning Integrations*.

Wettschereck, D., and Aha, D. W. 1995. Weighting features. In *Proceedings of the First International Conference on Case-Based Reasoning Research and Development*, 347–358. London, UK: Springer-Verlag.

Wooldridge, M. 2002. *An Introduction to Multiagent Systems*. New York, NY, USA: John Wiley & Sons, Inc.