Carleton University

95.495B – Honours Project

Applying Division of Labour Principles to Robot Soccer

Author: James Helferty, 232755

Advisor: Dr. Tony White, School of Computer Science

Date: April 28, 2003

Abstract

The goal of this project was to determine the feasibility of using the ACO specialization algorithm for in-play optimization of a robot soccer team. A robot soccer team written for TeamBots was adapted to use the algorithm, and tested and tuned against various opponents. Improvements were negligible, balanced by some of the negative consequences of the algorithm, which included moments of ineffectiveness due to travel between field positions, and the difficulty of the algorithm to react with decisiveness to changing field conditions.

Acknowledgements

I would like to thank Dr. Tony White, of the School of Computer Science at Carleton University, for his assistance and guidance in the preparation of this project and report.

I would also like to thank Tucker Balch, author of the TeamBots simulator, and M. Bernardine Dias, upon whose robot soccer team this project is based, as well as the authors of the various other soccer teams included with TeamBots, including Tucker Balch, Cristian Dima, Matthias Felber, David H. Johnson, Brian McNamara, Bryan Nagy, Mark Sibenac, Gita Sukthankar, and Håkan L Younes.

Table of Contents

List of Figures

1 Background Information

1.1 TeamBots

TeamBots was designed and developed by Tucker Balch at Carnegie Mellon University, as a development and testing environment for studying collaborative robot behaviour. Originally titled JavaBots, due to its being written entirely in Java, the simulator is designed to provide a rapid prototyping environment for multirobot logic development. [Balch and Ram, 1998]

The language Java was chosen for three main reasons; portability, productivity, and modularity. Java interpreters are available on many operating systems, including those employed on the testing robots themselves, making it highly portable. Likewise, the researchers observed that they produced working code much more quickly in Java than they did C or C++, increasing their productivity. Lastly, the highly OO-nature of Java allows them the modularity they desired to be able to reuse the different logic and sensor modules they had developed in each of their own projects [Balch and Ram, 1998].

TeamBots is not the only soccer simulation environment available for use. The official RoboCup Soccer Simulator is, in fact, the more popular and well known, owing to its use in the yearly RoboCup tournaments [Anon., 2003]. However, RCSS teams tend to be written in C/C++ and targeted at Linux [Cisternino and Heintz, 2003]. Since TeamBots was written entirely in Java, and the primary system for development was a Windows machine, TeamBots proved itself to be the more appealing choice.

TBSim is the title given to the TeamBots simulator. TBSim was designed such that it would simulate the operations of a variable number of robots on a field of arbitrary size. Obstacles and spherical objects are also simulated in 2D. A wall, for instance, is simulated as a line through which no robot may pass; a ball as an object capable of being kicked and pushed around, slowed by a constant decrease in velocity as it travels across the simulated ground.

The configuration of these objects is referred to as a Domain, which is read from a text file at the beginning of a run. Specified in this text file is a list of every object and obstacle in the Domain, along with their start positions and any other operating parameters necessary for initialization. Once a Domain is loaded into TBSim, a visual representation of the field appears, and simulation begins.

Due to the presence of Java interpreters as operating environments on actual robots, porting the logic of a TeamBots robot to hardware is, in theory, no more complex than the simple act of copying the file onto the robot. The interfaces provided for the robot logic in the simulator, TBSim, are identical to those offered in hardware, TBHard. There are no modifications that need to be made to the team's logic code whatsoever.

The interface provided by TeamBots from logic to hardware is completely abstract, and as such, standardized across all supported robot kits; adding basic support for simulation of a new robot is as simple as adding logic to four function calls; getHeading, getPosition, setSpeed, and setSteer. Additional functionality beyond the basics is

provided in the implementation of abstract Sensor and Actuator interfaces. [Balch and Ram, 1998]

TeamBots supports the Probotics Cye as part of Carnegie Mellon's Minnow project. [Stancliff, 2001]

1.2 Division of Labour

In social insects, the work of the colony is divided up into tasks. These tasks are then divided up amongst the individual insects in the colony. How they are divided up is a matter of some interest to AI researchers.

In nature, there are three main trends in the way tasks can be divided up in a colony [Bonabeau et al., 1999]:

1. *Temporal polyethism* – Individuals born in the colony around the same time period tend to perform similar tasks. It is not known if absolute aging is a factor, however, the social and external environment appear to influence behavioural development.
2. *Worker polyethism* – Individuals in different worker castes have different morphologies. The tendency is for workers within the same caste to tend towards similar types of tasks. Workers in different morphological castes tend to do different types of tasks.

3. *Individual variability* – Within a given caste, the individuals can develop heterogeneous behaviours, with some individuals tending more to working on one type of task than another. This is also referred to as a behavioural caste.

There is a certain level of plasticity in the examples of division of labour that occur in nature. Insect colonies have to be able to deal with factors such as perturbations in food availability, weather conditions, war, disease, etc. In nature, if a particular caste is wiped out, or its numbers reduced, the other castes will adapt their behaviour to take on the tasks previously performed by the other members. This is considered to be part of the division of labour problem.

1.3 Specialization

The algorithm Bonebeau [et al., 1999] suggests to model this division of labour is the specialization algorithm; it is designed to model behavioural castes. From an initially homogenous set of individuals, the result of the algorithm is to end up with a heterogeneous set of individuals, each member of which is specialized to a specific task.

In order to model this problem, each individual has a certain threshold for working on a task, as well as a stimulus for doing that task. The threshold lowers when they engage in that task (or learn it) and rises when they're not doing that task (forgetting it). Depending on the threshold value, the individual can have a greater or lessened probability of responding to the exact same level of stimulus.

The idea behind the algorithm is that individuals with more experience, and which are thus better equipped to handle a specific task, are more inclined to partake of that task than individuals who have less experience with that task.

The probability of an individual *i* undertaking a task *j* is expressed as:

$$T_{\theta ij}(s_j) = \frac{s_{i,j}^2}{s_{i,j}^2 + \alpha\theta_{i,j}^2 + \beta d_{i,j}^2}$$

Where $\theta_{i,j}$ is the self-reinforcing threshold for individual *i*, task *j*, and $d_{i,j}$ is the distance from individual *i* to where task *j* is performed. α and β are tuning coefficients, which we simply to 1. (We are already normalizing, and thus weighting, $\theta_{i,j}$ and $d_{i,j}$.)

Whenever individual *i* is performing task *j*, the self-reinforcing equation is:

$$\theta_{i,j} \leftarrow \theta_{i,j} - \xi\Delta t$$

Whenever individual *i* is not performing task *j*, the self-reinforcing equation is:

$$\theta_{i,j} \leftarrow \theta_{i,j} + \varphi\Delta t$$

The value of $\theta_{i,j}$ is restricted to between 0 and a maximum value. (We use 1)

For our implementation, we use values of $\xi = 0.00003$ and $\varphi = 0.00002$.

As the individual performs one task more than others, this causes the threshold for that task to drop, while the others increase. Since the probability function is based on the threshold, a lower threshold means a greater tendency to perform that task, reinforcing the selection of that task, thereby reinforcing the behaviour.

The use of a distance in the equation allows for us to give a higher probability to those individuals that are closer to the task performance location.

2 Implementation

The implementation consists of three main classes. The Specialization class provides an interface to the implementation, while the SpecializationConfig class provides instant access to configuration information. Internal to the implementation is the Task class, which represents a task within the context of the Specialization algorithm.

TeamBots allows one to specify a minimum frequency with which a robot will be given new information, and allowed to make decision changes in. By default, the simulator guarantees one of these cycles every $1/10^{th}$ of a simulated second. TeamBots refers to these occurrences as "TakeStep" intervals.

Each time the robot is instructed to take a step, we must update the Specialization algorithm as to what is going on. For the algorithm to work, we must keep it appraised of the current level of all stimuli.

In general, we calculate these values as follows:

$$s_j = \sum_k \left[w_k \cdot s_k \right]$$

We restrict the values of the constant weight coefficients, w_k, such that

$$\sum_k w_k = 1$$

2.1 Basis Team

The team that the specialization algorithm was integrated into was called FemmeBotsHeteroG, and was originally written by M. Bernardine Dias of Carnegie Mellon University. This team was selected for its simplicity; the team has one "center," two "attackers," a "defender" and a "goalkeeper."

The "center" hangs around the centre of the field, waiting until the ball comes within its sphere of influence. Then it heads straight for the ball. Once it is the closest one to the ball, the centre tries to get it as close as it can to the opponent's net, and potentially score. If it is no longer the closest one on its team to the ball, it returns to its wait position in the centre of the field.

The "attacker" waits around centre while the ball is in its own end, and attempts to score while the ball is in its opponent's end. There are two attackers, the north and the south. The only difference between the two is where on the field they wait, the north attacker stays just north of centre, while the south stays just south of centre. When attempting to score, the two work together to get the ball into the opponent's net. If they are unable to get it in by kicking, they work together, and use their brute strength to push any defender out of the way as they take the ball to the goal.

The "defender" waits near the goalkeeper for an opponent to enter the two-thirds of the field closest to its team's net. The defender then rushes to get into position between the ball and its net. If the defender is the closest one to the ball, and it has a clear shot, it

kicks the ball towards the opponent's goal. If the ball travels into one of the close corners, it is the defender's job to brace against the goalkeeper so the ball can't be pushed into the net.

The "goalkeeper" waits at the net for the ball, tracking its movements back and forth (north and south) across the field, as far as the net goes. As the ball draws closer to the net, the goalkeeper continues to track the ball, attempting to punt it away from the net as it comes within range. Should the goalkeeper find itself out of the net, perhaps as a result of being pushed by an opposing player, it attempts to return to position as quickly as possible.

Since a goalkeeper is deemed necessary regardless of perceived demand, much like an insect queen to a nest, the goalkeeper is considered independently of the specialization algorithm.

2.2 Stimulus Calculations

The stimulus equations given below are not entirely correct; we must ensure that the values of the stimuli stay within bounds at all times (0 to 1). Hence, before being subtracted from 1 and multiplied by their scaling factor s_k, each equation is corrected to maintain bounds; if it is less than 0, s_k is set to 0. If the equation's result is greater than 1, s_k is set to 1. (This step is not shown in the notation.)

The stimulus values used in the implementation are as follows:

$$s_{center} = 0.5 \cdot \left[1 - \frac{|v\_ball\_to\_center|}{F\_DIAG \cdot 0.5} \right] + 0.5 \cdot \left[\frac{\sum_{t}^{teammates} [F\_DIAG \cdot 0.5 - v\_center_t]}{F\_DIAG \cdot 0.5} \right]$$

$$d_{center} = \frac{|v\_to\_center|}{F\_DIAG \cdot 0.5}$$

v_ball_to_center is a vector from the ball to the centre of the field. *F_DIAG* is a constant specifying the diagonal distance from one corner of the field to the other. v_center_t is a vector from team mate *t* to the centre of the field.

$$s_{attack} = 2 - \frac{|v\_ball\_to\_goal|}{F\_DIAG \cdot 0.5}$$

$$d_{attack} = \frac{|v\_ball\_to\_me|}{F\_DIAG}$$

v_ball_to_goal is a vector from the ball to the opponent's goal. *v_ball_to_me* is a vector from the ball to the robot. This stimulus equation turns 1 whenever the ball crosses the line into the opponent's territory, and gradually fades linearly to 0 as the ball travels closer to the home goal.

Since the north and south attackers are essentially the same, we treat them as one role. When assigning a robot to this role, we randomly switch between which to assign them to. This has the side effect of causing attackers to never truly settle into a position at the

centre of the field, instead preferring to jump back and forth from the north rest position to the south.

$$s_{defender} = 0.5 \cdot \left[1 - \frac{|v\_ball\_to\_our\_goal|}{F\_DIAG} \right] + 0.5 \cdot \left[\frac{\sum_{l}^{opponents} [F\_DIAG - v\_net_l]}{\sum_{l}^{teammates} [F\_DIAG - v\_net_l]} \right]$$

$$d_{defender} = \frac{|v\_def\_zone|}{F\_DIAG}$$

v_ball_to_our_goal is a vector from the ball to the team's home goal. *v_def_zone* is a vector from the robot to the position where the defenders play. (In this case, the net.) v_net_l is a vector from robot *l* to the team's home goal.

The summations of the opponents and teammates are subtracted from eachother to obtain a value that corresponds to the balance of defenders to the number of our opposition's attackers in the defensive zone. If there are more opponents in our end than there are teammates, this should spur the creation of more defenders to compensate.

2.3 Task Decisions

For each step, the stimulus is updated, and the specialization routine is checked to determine which role the player should be engaging in.

First, the routine checks if it is already engaged in a task. If it is, the routine checks to see if it has exceeded the minimum interval, 1/*p*, before dropping a task. If it has, the task is dropped.

We use a value of $p = 0.0005$. (Making the minimum interval 2000ms.)

Next, the routine checks again to see if it is engaged in a task. If it isn't, then the routine goes through the process of selecting a new task. The process of determining a new task is equivalent to taking the probabilities of all of the possible tasks, and putting them on a number line. Then a random number *r* is rolled, and wherever that number falls on the number line is what the new task will be.

This is implemented as follows:

$$t = r \cdot \left[\sum_j \left[T_{\theta ij} \right] + T_{\theta_{idle}} \right]$$

where *t* is the target value we'll be looking for, and $T_{\theta_{idle}}$ is the idle probability constant. (Our implementation uses a value of 0.) We then loop through the tasks, adding up their probabilities, until we reach the task where the value of the thus-summed probabilities exceeds *t*. This is the task we select.

If we exceed the list of probabilities, then the target task is obviously the idle task. In this eventuality, we select no role for the individual *i*.

We repeat this process for as long as the robot is functioning.

3 Observations

3.1 Description of Test Environment

The testing environment was:

- AMD Duron 2100+, operating at 1314MHz
- 384 MB
- Windows 2000
- Java 2, v.1.4.1
- TeamBots 2.0e

3.2 The Opponents

All robot teams were tested using the simulator from the TeamBots 2.0e package. The description file used by the simulator was the exact same as the one supplied, excepting the portion pertaining to the class files of the robots to be used. SpFemmeBotsHeteroG (our specialization implementation of the FemmeBotsHeteroG team) was supplied in place of the name of the default team for the west side. The following robot teams were tested against as control teams on the east side:

- FemmeBotsHeteroG
- DaveHeteroG
- AIKHomoG
- BrianTeam
- CDTeamHetero
- SchemaDemo
- MattiHetero

What follows are general descriptions of the strategies employed by each of these teams.

3.2.1 DaveHeteroG

The DaveHeteroG team follows this simple tactic; figure out who on each team is closest to the ball, and then match up the rest of our players to block theirs. For this strategy to work, three classes of individuals are employed, the "goalie", "offender", and general purpose.

The "goalie" acts as the team's goalkeeper, staying flush against the goal, and attempting to stay at the same position north and south as the ball. Should the ball's position exceed the limits of the goal, the goalie stops at the corner of the goal it's tending. Of note is the fact that the goalie is not a set individual; in the event that another robot gets closer to the goal than the current goalie, they become the new goalie. This is highly effective in combating teams that employ a strategy of pinning the goalkeeper away from the net.

The "offender" constantly tries to get behind the ball and kick it towards the opponent's goal.

The rest of the team's players are general purpose. Their strategy is to match up against the opposition's players, each one taking on the closest opponent to them not already taken. They then attempt to block these players from doing their jobs by pushing them

away from everyone else, in a scattering attempt. Should one of these players get too close to the goal, they will swap in and become the new goalie.

3.2.2 AIKHomoG

This is a homogenous robot soccer team, written by Håkan L Younes. All individuals behave using the same logic.

If the robot is not the closest to the ball, a force calculation is made. Negative forces are added for teammates and walls, and a positive force is added for the opponent goal's position. Lastly, a positive force is added for the two offensive positions just outside the corners of the opposition's goal.

A check is made to see if there is already someone tending goal, and if there is not, then the robot heads back to perform this task. Otherwise, the robot travels in the direction indicated by the force calculation.

If the robot is the closest to the ball, then the robot calculates the best position to kick the ball from, goes there, and kicks it.

3.2.3 BrianTeam

This is another homogenous team, written by Brian McNamara. The strategy is documented in the source code as follows:

```
if( i have the ball ) then
   if( clear shot ) then
      shoot
   else if( i am in ourgoal ) then
      kick away ball
   else
      go towards center
   endif
else
   if( i am closest to ball ) then
      go to "behind" ball (avoid bumping it backwards)
   else
      if( i am closest to ourgoal ) then
         go to point between ball and ourgoal nearest ourgoal
      else if( close enough to nearest opponent )
         mark a man (go to nearest open opponent)
      else
         get open (move away from nearest opponent)
      endif
   endif
endif
```

3.2.4 CDTeamHetero

This soccer team was written by Cristian Dima, and is based on an example written by Tucker Balch.

If the robot is closest to the ball, it attempts to kick the ball into the opponent's net. Otherwise, it behaves according to its role on the field, "forward," "assistant forward," "blocker," "defender" or "goalkeeper."

If the ball is in its team's end, the "forward" waits at centre for the ball to come out; then it heads to its kicking location just outside the opponents net. The "assistant forward" does the same, only it hangs back slightly from the forward.

The "blocker," "defender" and "goalkeeper" all employ the same strategy; keep their bodies in between the ball and the centre of the net. The blocker does it at three goal widths out, the defender at two, and the goalkeeper does it from flush against the net.

3.2.5 SchemaDemo

This is a simple team written by Tucker Balch using the Clay robot modelling package which comes with TeamBots. The Clay package works on a system of pre-defined schemas, weighted together to yield a decision result.

There are two roles available to the SchemaDemo team, the "goalie" and "forward."

The "goalie" has two embedded steering schemas; get behind the ball, and move to the back of the field. The get behind the ball schema is primarily a swirl around the ball schema, weighted slight more than a move to halfway between the ball and our goal schema, and an avoid bumping into teammates schema. The move to back of the field schema is a move to goal schema, as well as to a lesser extent a move to ball schema.

The “forward” also has two embedded steering schemas; get behind the ball, which is the same as the one for the goalie, and a move to ball schema. The move to ball schema is primarily a move to sweet spot near goal schema, with a slightly lesser degree of the avoid teammates schema.

Whether or not the ball is kicked, in either of these roles, is determined by a kick ball schema built into Clay.

3.2.6 MattiHetero

Written by Mattthias Felber, and originally based on BasicTeam, this is a heterogeneous team with four roles; “left wing,” “right wing,” “back wing,” and “goalie.” All behave the exact same way if they possess the ball; they attempt to get in close enough to get a clear shot at the opponent’s net, and then they take it.

The “goalie” follows the ball back and forth on the field, staying within a defensive radius of its home goal. Should the ball go further north or south of another player on the field, the goalie stays inside of that player’s position, so as to avoid preventing them from becoming a left- or right-winger. If the ball is in close, the goalie stays close to the net; if the ball is out a fair distance, the goalie comes out, so as to avoid any goalie interference tactics presented by the opposing team.

“Left wing” and “right wing” players stay behind the ball. Should it veer too much to one side, they serve to guide it back towards the centre of the field, and keep it from going out of play.

“Back wing” players avoid their closest teammate, and attempt to stay behind the ball.

3.3 Interesting Behaviours

While testing against the various control teams, various trends in behaviour were observed. What follows is a list of these observations, along with examinations of their potential causes.

3.3.1 Centers

One of the most effective strategies of the specialization team was made possible through the exploitation of a deadlock-preventing feature built into the simulator. Whenever the ball's position is frozen for several seconds, a timer will run down and then the ball will be dropped again at the centre of the field. To exploit this behaviour of the simulator, the author of the original team coded the center so it would hover around the middle of the field, waiting for these ball drops. Attackers were also coded to hover in the same area when not in use, so as to be prepared to run in with the ball after a drop.

Freezing a ball is a relatively simple task. Since all that is required for the ball to be considered frozen is for its position to remain constant for a short period of time, there are two manners by which this can be made to occur; the ball can either be kicked into a position where it takes too long for a player to get to it before it times out, or the ball can be trapped between a player and another object on the field.

An effective strategy for defensive robots then, in removing the ball from their zone, is to either kick the ball into the corner, where it is unrecoverable (either through time restrictions in a robot travelling to it while it's at rest, or in its being left flush against the wall), or to push back with equal force against an oncoming opponent intent on scoring. If done effectively, the latter will cause the ball to become stuck between the two robots, and shortly thereafter, dropped in the centre of the field. A center is then able to quickly gain possession of the ball, and charge towards the opposition's goal.

In the specialization team, it was important to maintain the effectiveness of this tactic. By including a center-generating weighting value in the specialization equations in the specialization team, we ensure there remains an impetus on the team to leave at least one robot in the center role. As a result, it is relatively rare that the centre of the field not have a robot nearby engaging, or prepared to engage, in center role behaviour.

However, there are still times we can end up with an absence of players near the center. Fig. 1 shows a game against DaveHeteroG where the centre of the field has been left unprotected. The center that had been occupying that area had rushed back to take on a defender role temporarily, leaving no robots available to take ball possession following the drop. While in Fig. 2, the defender is able to make it to the ball before the others, and keep it inside the opposition's zone, we have lost field position as the other team has the time to get into a better defensive position in the middle of their zone.

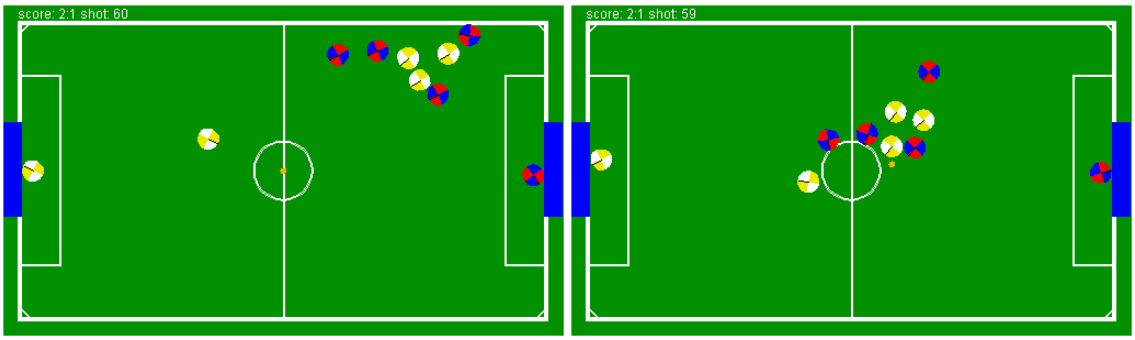

FIGURE 1 – (left) SpFemmeBotsHeteroG (light) versus DaveHeteroG (dark). No center; a defender comes forward to fill the void.
FIGURE 2 – (right) Having kicked the ball forward towards its teammates, the same robot turns to resume its duties as defender near the net.

This occurs because even when the strength of the defender stimulus is low, if the threshold for a defender is low enough, there is still the random possibility that our robot will assume the role, regardless of demand. The defender stimulus drops whenever performing defender behaviour, which is sparked by insurgences by the other team into our zone. Hence, the probability of centers spontaneously becoming defenders is proportional to the probability of the opposing team having been deep in our zone sometime recently.

As we can see from the threshold graphs (Fig. 3-6), we ended up with no dedicated centers this game. However, from the various dips from all players, we can see that performance of this task was split relatively evenly amongst all involved. The exception is player two, which was close to becoming a dedicated center at one point. This coincides with a strong offensive by our team; even though player two was an attacker, it was the closest one to the centre of the field, and so it took on the role of center. The following return to previous levels was sparked by the center receiving possession of the

ball, and bringing it into the opposition's zone. The attacker stimulus then overrode the center stimulus, and player 2 returned to being an attacker.

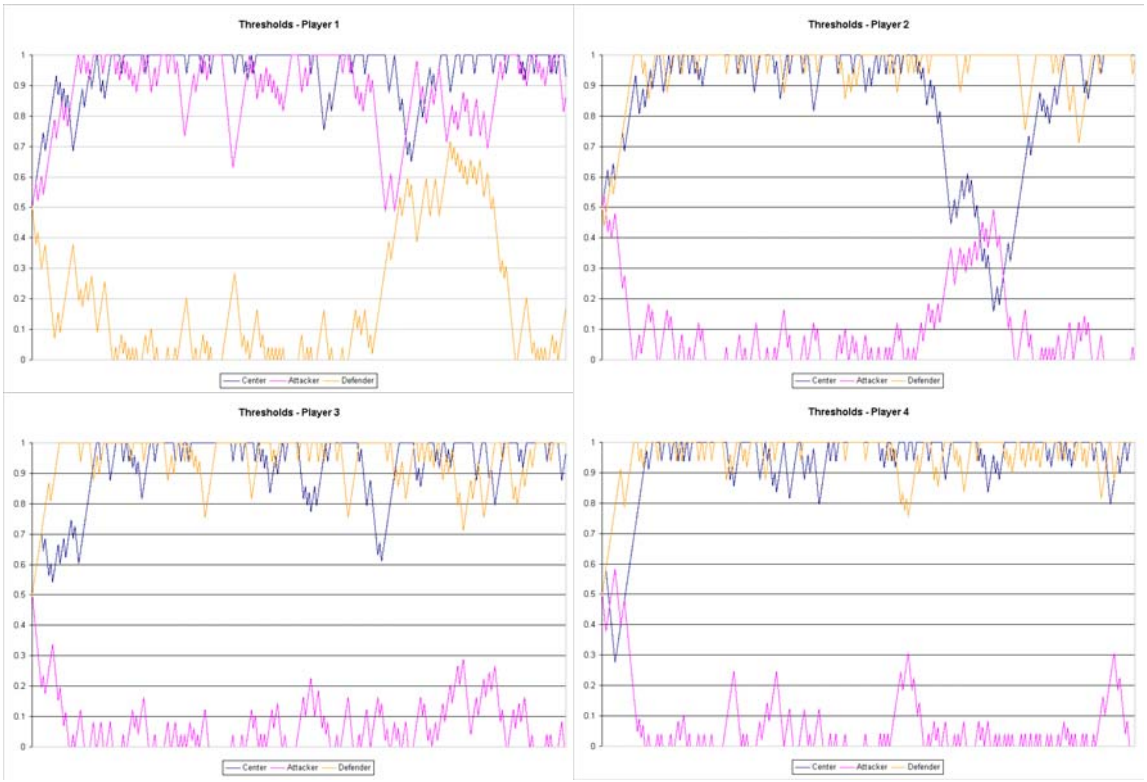

FIGURE 3 – (top left) Thresholds – Player 1. Primarily a defender; the downward spikes at the end for center indicate performance of the center behaviour, when the defender rushed forward to kick the ball to its teammates.
FIGURE 4 – (top right) Thresholds – Player 2. Primarily an attacker.
FIGURE 5 – (bottom left) Thresholds – Player 3. Primarily an attacker.
FIGURE 6 – (bottom right) Thresholds – Player 4. Primarily an attacker.

Curiously, the degree of specialization towards being a center in this case is indicative of the whole of the experiments run. Center specialization with this set of stimulus equations is exceptionally rare. Increasing the stimulus levels for the center relative to the others will lead to increased specialization as centers, and better odds of collection after a ball drop. However, the trade off is in effectiveness of defense, and the commitment of attackers, whose memberships tend to suffer as a result.

With more centers, the game becomes more back and forth, with an incursion into one zone, immediately followed by an incursion into the other. Defenders are not necessary with such tactics; for such a team, it becomes a matter of grabbing the ball as soon as it appears, rushing to get it close to the net, and then either making a lucky shot, or pushing their goalkeeper sideways until the ball falls into the net. So while a stronger center specialization does help the team win more games, it is by relying on the strengths of our team's goalkeeper, and the lack of exploitation of the same brute-force scoring techniques by the other team, rather than on the specialization algorithm itself.

The other flaw of having a more powerful center stimulus is due to the overlap in job description between a center and an attacker. A stronger center stimulus would have to be accompanied by a tighter description contributing to the stimulus; simply increasing the stimulus amount can lead to confusion on whether to perform a center or an attacker role. This confusion can manifest itself in a lack of commitment on the part of an attacker, who will turn away back towards centre while they are still relatively close to the ball, and deep and alone in the opposition's zone.

By having a weaker center stimulus, we are still able to generate a sufficient minimal number of centers because our attackers loiter in the same position. Thus, when the ball is dropped, they are close enough that the proximity will cause the center stimulus to override the attacker.

3.3.2 Task Changes

There are five players on each team. On the specialized team, four of these can become specialized in a particular task, of which we have three.

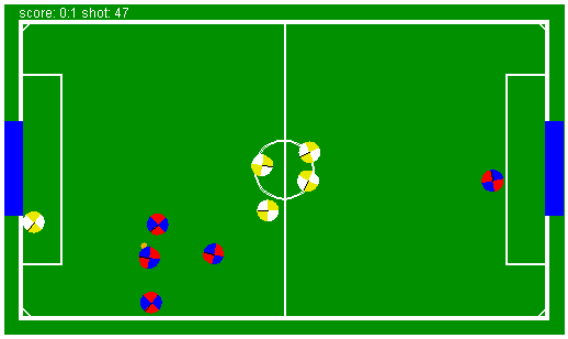

FIGURE 7 – SpFemmeBotsHeteroG versus SchemaDemo; all of the specialization individuals have become attackers.

In this instance, we have had all four individuals on our team choose the attacker role at the same time, in response to the ball being in our opposition's zone. Shortly thereafter, the ball passes across the line, and moves into our zone, with all four of their attackers accompanying the ball.

Our attacker role does not specify any sort of defensive behaviour, leaving our goalkeeper alone to defend against the incursion. Even if several of our specializing players manage to become defenders, it may already be too late, as it would still take at least five seconds for the closest one to reach the goal.

Statistically, all of our specialization players choosing to be an attacker when we need defenders is rare. However, there are other more common scenarios with similar consequences.

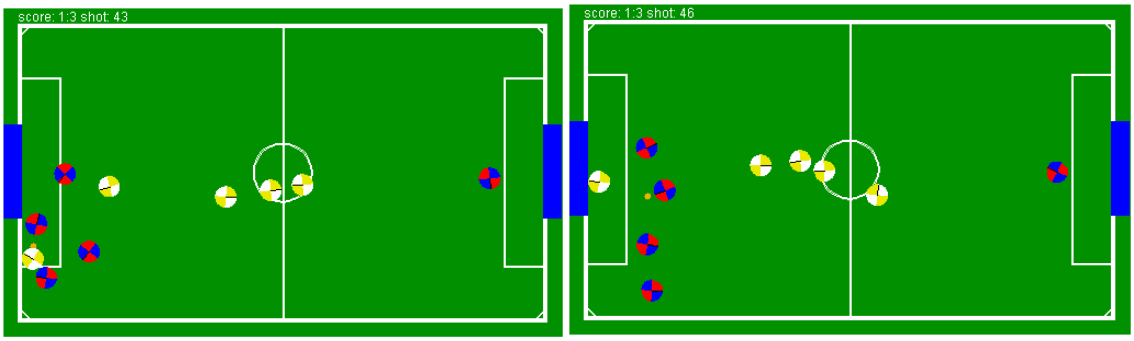

FIGURE 8 – (left) SpFemmeBotsHeteroG (light) versus SchemaDemo (dark); two defenders, two attackers
FIGURE 9 – (right) SchemaDemo preparing to score; two defenders, two attackers (two have swapped roles)

Fig. 8 illustrates a game between our team and SchemaDemo. The SchemaDemo team has made it into our zone, and the ball is between our goalkeeper and an opposing player and the net. Of our specialized players, there are currently two defenders, and two attackers. One defender is over halfway to the goal.

Fig. 9, however, shows what happens shortly thereafter. The closest defender has changed roles to an attacker, due to its low attacker threshold. At the same time, one of the attackers has changed roles to defender, due to the low ratio of defenders to opponents in our zone. While we now still have the same number of attackers and defenders, we have lost field position in a critical situation. The other team scores shortly hereafter.

This occurs because all of the players on our team have developed low attacker thresholds. If we spend a substantial amount of time in the other team's zone, then we will tend to have more dedicated attackers than defenders. This will happen because the stimulus equations are influenced by the amount of time spent performing the behaviour.

Should the opposing team be able to dip in quickly, and score effectively each time they do, our team will be completely unprepared to defend itself against these attacks.

In this instance, even though all of the players on the opposing team are deep within our zone, and are preparing to score, our players still have a higher tendency to want to assume the attacker role than the defender role, due to their defender thresholds being substantially higher.

In an insect colony, there can be thousands of individuals available to do the work of the colony, while in a robot soccer team there are only four. Since the proportion of individuals to tasks to be performed is substantially lower, it stands to reason that indecisive behaviour such as this could be more apparent a detriment to the efficiency of this algorithm in a soccer team.

3.3.3 Goalkeeper Interference

An effective, and allowed, tactic in the TeamBots simulator is to prevent the goalkeeper from doing their job, by restricting their movement in front of the net. There are several teams included with the simulator that intentionally exploit this tactic. (eg; Kezche, JunTeamHeteroG, etc.) Since the basis team was ill equipped to defend against these tactics, they were dropped from testing.

In practice, however, a goalkeeper interference tactic may end up being unintentionally leveraged by an aggressive opponent. Fig. 10 demonstrates a scene from a game between our specialization team and its basis team. In this instance, the goalkeeper is being prevented from doing its job because a defender is in its way, which is in turn being blocked by an opposing player. (This can also happen if we have more than one defender, and they are both fighting for the same position.)

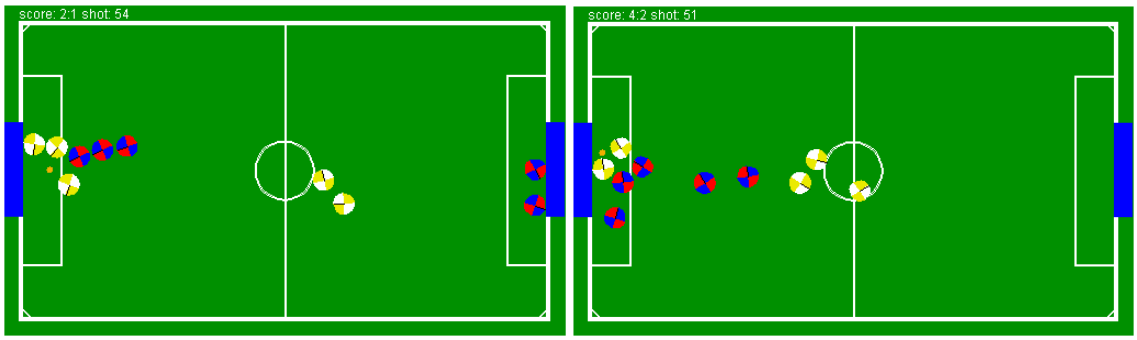

FIGURE 10 – (left); SpFemmeBotsHeteroG (light) versus FemmeBotsHeteroG (dark); the goalkeeper, blocked by defender, blocked by attacker from the other team, as the ball trickles into our net.
FIGURE 11 – (right); SpFemmeBotsHeteroG (light) versus MattiHetero (dark); the goalkeeper is blocked by a defender which has just turned into an attacker.

Unwise specialization decisions in our own team can be just as problematic. Fig. 11 shows a scenario where the opposing team is about to score because our goalkeeper is being blocked. In this instance, it is the fault of the player in front of the net, which has just changed to the attacker role. The attacker makes no attempts at avoiding the ball, or at giving way to the goalkeeper as it turns to pursue its new role at the centre of the field.

Also visible in Fig. 11 is an effective tactic used by the MattiHetero team (and also by AIKHomoG) in preventing goalkeeper interference tactics from being used against its team; the goalkeeper does not remain against the net, but rather stays out from the net. In

this way, if an opposing player attempts to push them out of the way, they will still have the ability to pull back and around them, and into position.

3.3.4 Defensive Play Against Aggressive Opponents

If the team is performing particularly badly, and the other team is in our zone for a large part of the game, this can inspire more team members to become defenders. Fig. 12-15 show the resultant threshold values of play against AIKHomoG. This team is exceptionally aggressive, and spends most of its time in our zone. Our team attempts to balance against this configuration.

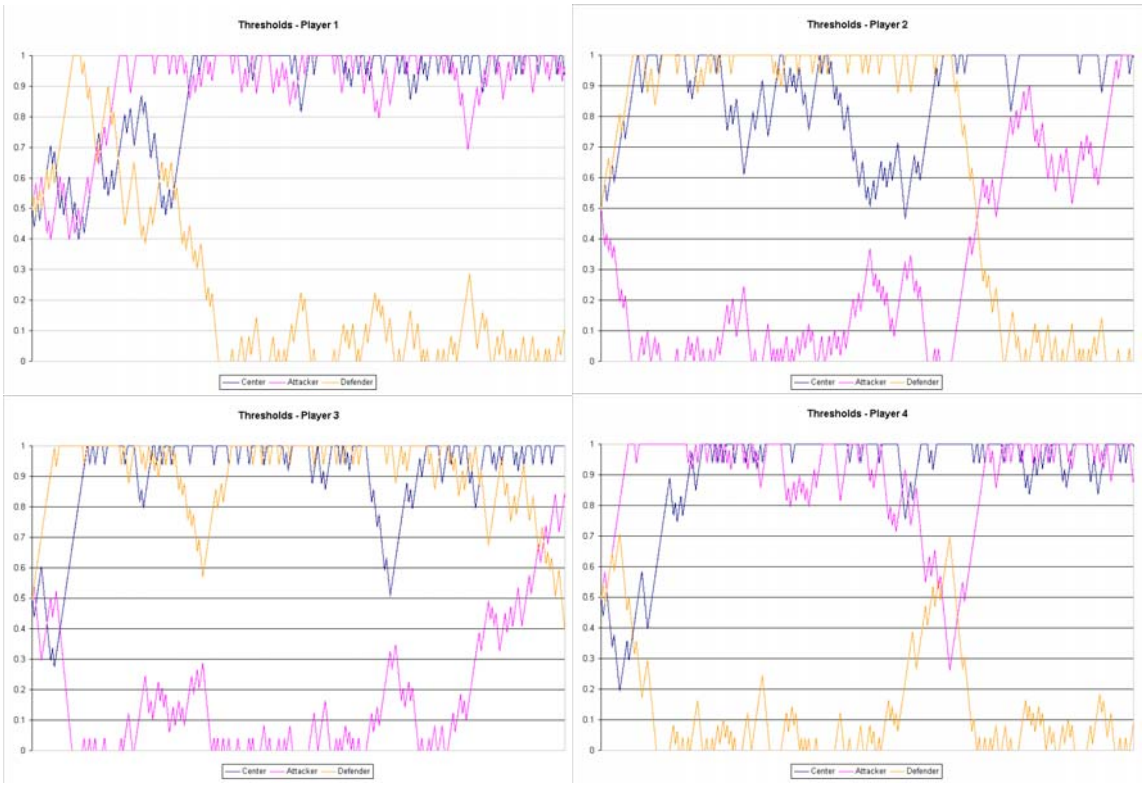

FIGURE 12 – (upper left) Thresholds – Player 1; SpFemmeBotsHeteroG versus AIKHomoG.
FIGURE 13 – (upper right) Thresholds – Player 2
FIGURE 14 – (bottom left) Thresholds – Player 3
FIGURE 15 – (bottom right) Thresholds – Player 4

As we can see from the graphs, player four becomes a defender almost immediately, followed by player one and player two. As the game ends, we can see player three becoming a defender as well.

The key to this opponent's success is in the two players it leaves just outside the corners of our net, whose only purpose is to score. This team is purely an offensive team, which makes them a difficult competitor for the specialization team. On the other hand, their defence is relatively weak. The center-heavy configuration mentioned in 3.3.1 put in a better showing against this opponent for just this reason; when attacked en masse, AIKHomoG's lone goalkeeper puts up little resistance.

3.3.5 Delay of Game

The ball drop on ball freeze feature of the simulator can further be exploited for delay of game. Should the ball ever be re-dropped in the middle of three robots ringed around the centre, the ball is essentially frozen permanently, and whomever was winning up until that point will win the game.

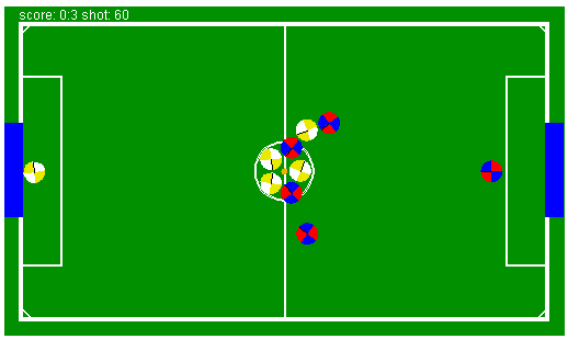

FIGURE 16 – SpFemmeBotsHeteroG (left) versus SchemaDemo (right); the ball is frozen for the rest of the games, as the two teams refuse to yield, and allow the other team room to knock the ball from the centre after a ball drop.

While no teams were observed to actively exploit this flaw in the simulator, it is conceivable that a team could be designed, or possibly evolved through a GA, to exploit this flaw in the simulator, freezing the ball in the centre of the field while they are in the lead and ensuring their victory.

4 Conclusions

The implementation presented demonstrates several fundamental strengths and weaknesses in applying the specialization division of labour algorithm to a simulated robot soccer team.

The time spent travelling between roles can be an impediment to the performance of the team. Since the roles in the basis team perform their behaviours in different general areas on the field, transit time from one area to another after a role decision becomes a factor.

This problem can be compounded if we accidentally choose a poor role, which does happen quite frequently. With this algorithm, position on the field does not matter to an individual to the same degree as what role the robot was engaged in most recently for the longest period of time. As such, players may miss good opportunities because they are more inclined to perform the roles they're more comfortable with.

One possible improvement to fix this problem would be a further refinement of the stimuli for the various roles on the field. For example, a dampening value might be subtracted from the defender stimulus when the player is currently in possession of the ball, and close to the opposition's goal. The challenge of such a refinement would be in ensuring a balanced set of dampening values for all roles, so that one role doesn't get dampened more harshly or leniently than the rest.

The basis team tended to be hardwired for its tasks with a specific number of robots (one) per role in mind. As such, it did not scale well to a varying number of individuals in the defender position, for instance, where extra defenders tended to get tangled up in each other.

In this implementation, our tuning coefficients α and β for our probability equation were simplified to 1. A further refinement of this implementation may be to experiment with these values, to see what effect modification of them might have on the ranges at which individuals participate in team behaviour.

Furthermore, a team balancer stimulus might be employed to give the team a more aggressive or defensive posture. (This global team reinforcement strategy has been suggested and employed before by Balch [1997].) If the team is losing, then it may make sense for the defender's stimulus to be added to a reinforcement value, which would spur the creation of defenders even whenever there are no attackers in our half of the field.

Likewise, if the opposition is spending a large portion of time in our half of the field, then perhaps we aren't being aggressive enough, and we need more centers to hold the line when the ball reappears in the middle. In situations like this, an aggression stimulus could be calculated and added to the regular center stimulus, to react to a perceived need for more centers.

The key strength of the specialization algorithm is that it is capable of adapting to changing conditions on the field. If we need more defenders, this algorithm remembers that fact on an individual-by-individual basis. Even if the ball goes into our opposition's zone, an individual will remember, for some time after that, to stay back so as to be prepared. In this respect, the specialization team performed its task as expected.

5 References

Balch, T., Ram, A. (1998) Integrating Robotic Technologies with JavaBots, Working Notes of the AAAI 1998 Spring Symposium, Georgia Institute of Technology

Anon. (2003) The RoboCup Soccer Simulator, http://sserver.sourceforge.net/

Cisternino, A., Heintz, F. (2003) The RoboCup Simulator Team Repository, http://medialab.di.unipi.it/Project/Robocup/pub/

Stancliff, S. (2001) The Minnow Project, http://www-2.cs.cmu.edu/~coral/minnow/

Bonabeau, E., Dorigo, M., Theraulaz, G. (1999) Swarm Intelligence, Oxford University Press, pp. 110-143

Balch, T. (1997) Learning Roles: Behavioral Diversity in Robot Teams, GIT-CC-97-12, Georgia Institute of Technology