SHEEPDOG TRIALS

PROJECT 4

Programming and Problem Solving

Fall 2013 / Professor Ken Ross

GROUP 5

Sam Aronson

Andrew Goldin

Mahd Tauseef

December 9, 2013

TABLE OF CONTENTS

1 PROBLEM STATEMENT

The goal of this project was to program an agent (or player) that assumes the role of a sheepdog, and must herd sheep across a fence, either alone or in a group of identical dogs. The field of play is a large square field with each side of length 100 meters, divided vertically in half (at x = 50m) by a fence with a 2m wide gate at the center. At the beginning of each trial, a number of sheep S are randomly and uniformly distributed on one side of the fence, with some number $B \leq S$ of them being black sheep and the rest white. A certain number of sheepdogs d start in the empty half of the field, each one executing an instance of the same code instantiated with a unique dog-id parameter (between 1 and d). The dogs are spread out uniformly along the far fence, meaning each starts at an initial position of:

$$pos = \left(0, \frac{id_{dog} * 100}{d + 1}\right)$$

Each dog has access to the position of every dog and every sheep, but they cannot explicitly communicate with one another. When the trial starts, the dogs will make their way to the other side of the field and proceed to herd sheep across to the other side (see Figure 1 below).

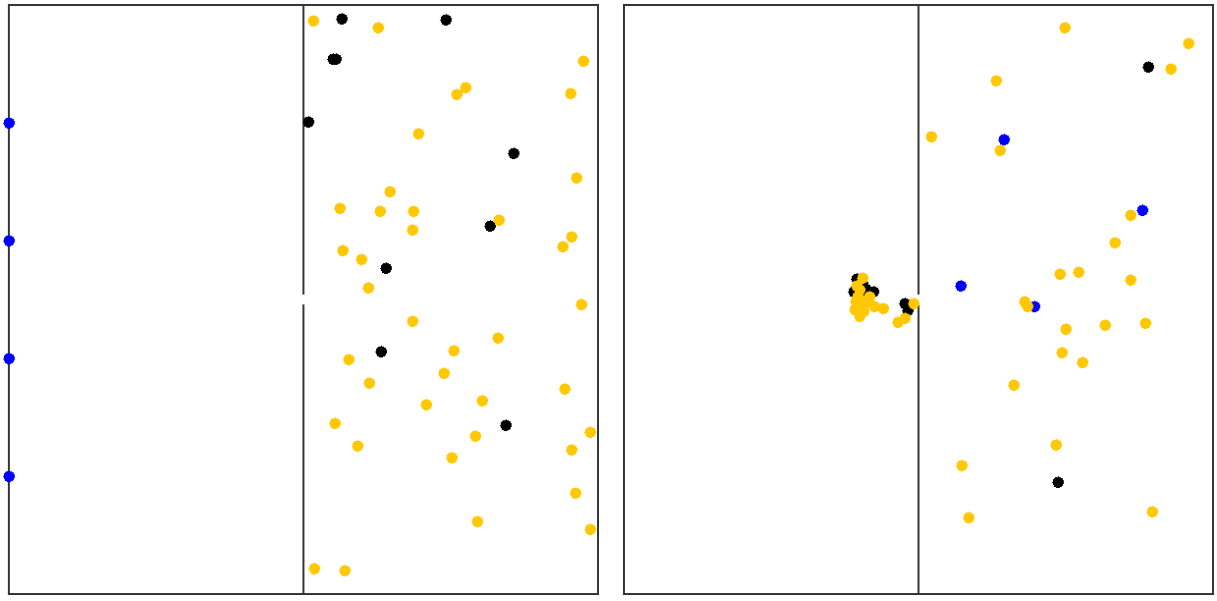

Figure 1: To the left, the simulator at start time, with four dogs (blue dots) lined up at the far fence of the empty half of the field, and 50 sheep (10 of them black) positioned in the other half. To the right is the simulator in mid-play of basic mode, with each dog using a one-by-one herding strategy.

Sheep react to dogs in the following ways. When a dog is between 2m and 10m from a sheep, the sheep will walk at a speed of 1m/s directly away from the closest dog. If a dog is within 2m of a sheep, the sheep will run at 10m/s away from the closest dog. The maximum speed of a dog is 20m/s. If a sheep hits a fence, it will bounce off as if it were a ray of light (angle of incidence equals angle of reflection). Sheep can also react somewhat to the presence of other sheep. Sheep that are

within 1m of one another will move away from each other at a speed of 0.5m/s. For sheep in a crowd, the direction each sheep will move is determined by the sum of unit vectors from each nearby sheep (within 1m). If a dog is also nearby, a net velocity is computed as the sum of the two velocity vectors. Dogs operate independently and do not exert any physical influence on one another.

The simulator itself operates discretely rather than continuously, with a granularity of 0.1s. This means that the maximum distance a dog can travel within one simulation timestep is 2m, and the position of each sheep at the next timestep is computed based on the positions of the sheep and dogs at the current timestep. At each timestep, the player will return a position in x, y coordinates that the dog should move to. It must be within 2m of the current position.

The player needs to be able to perform two tasks, basic and advanced. The basic task is to use the dog(s) to herd all of the sheep through the gate to the other side of the field. The advanced task is to only move the black sheep, while trying to keep the white sheep in the original half of the field. The performance measure for each task is simply based on the number of simulator ticks it takes to complete the task. The fewer, the better. If the dog(s) cannot perform the task within a certain maximum time allotted, the simulator will timeout and return a time of (1 + *maxTime*) ticks. Our group's goal was to create a player with a simple and robust strategy that would guarantee the completion of the task in a reasonable amount of time under realistic scenarios (e.g. the number of dogs d is proportional to the number of sheep S).

2 STRATEGY

Our player implements two distinct strategies, a one-by-one strategy and a convex hull traversal strategy. In the basic mode, the dogs will use a convex hull strategy if the ratio of sheep to dogs is above a certain threshold (explained further in later sections) and a one-by-one strategy otherwise. In the advanced mode, the dogs will utilize a special variant of the one-by-one strategy designed to return any white sheep that accidentally make it across the fence in the herding process.

2.1 ONE-BY-ONE

For the basic mode, the one-by-one strategy follows a simple two-phase system. To make movement simpler to program, our Player implements a function called *moveDog* that takes as arguments the current location, the desired location, and a speed between 0 and 2, computes the unit vector between the two points and scales it by the speed value to compute the exact next point the dog should move to. If the desired point is within 2m of the current point, the dog will move exactly to that point. The phases are as follows:

- **Phase 0:** Move the dog towards the gate. When the dog reaches the gate, switch to Phase 1a.
- **Phase 1a:** Choose a random, undelivered, unpursued sheep and switch to Phase 1b.
- **Phase 1b:** Travel straight to the chosen sheep and push it directly at the gate. When the sheep is delivered, return back to Phase 1a. Continue this cycle until all sheep are delivered and the simulator stops.

Phase 0 is very simple. Simply call *moveDog(current, center, maxspeed)* at each tick, where *current* is the coordinates of the dog's current position, *center* is the coordinates of the gate (50, 50) and *maxspeed* is 2 meters per tick. Once *current* = *center*, phase 0 is complete.

Phase 1 is divided into two parts: choosing a sheep and pursuing it. Phase 1a states for the dog to choose a random, undelivered, unpursued sheep. This is done with a two-step process:

1. Compute the set of undelivered sheep by creating a list of all sheep with an x-coordinate greater than 50. This implies that the sheep are still in the original half of the field.
2. Choose a random sheep from this set. If the sheep is already being pursued by another dog, continue to choose another random sheep until an unpursued one is found. If all sheep are detected as being pursued, "merge" with another dog in order to minimize unneeded influence on other sheep.

A dog marks a sheep as being pursued if, for a given sheep, there is another dog that is closer to that sheep, whose coordinates lie exactly along the vector (within some small floating point error) from the gate to the sheep. From this, it is reasonable to infer that the dog is chasing the sheep.

For Phase 1b, our player simply computes the unit vector $\boldsymbol{v}$ from the gate to the chosen sheep and at each tick moves towards the point $\boldsymbol{s} + \boldsymbol{v}$, where $\boldsymbol{s}$ is the current position of the chosen sheep. By the nature of this movement, once the dog makes it to the point such that the sheep lies between it and the gate, it will always be positioned (or moving towards the position) behind the sheep, exactly aligned with the gate. Since the sheep will move in the direction directly opposite the dog, it will naturally be pushed straight to the gate. Through testing, we found that this follow distance of 1 meter was large enough such that the dog can quickly and easily get behind the sheep, but small enough that the dog can keep the sheep running at its maximum speed towards the gate.

For the advanced mode, our player uses the same strategy detailed above, with a few small modifications. In this mode, our player needed a failsafe for returning any white sheep that accidentally make it to the other side of the fence. Additionally, if any black sheep make it back to the original side in the process of returning the white sheep, they need to be pushed back. This problem called for a cyclic process, defined below:

- **Phase 0:** Move the dog towards the gate. When the dog reaches the gate, switch to Phase 1a.
- **Phase 1a:** Choose a random, undelivered, unpursued ***black*** sheep and switch to Phase 1b.
- **Phase 1b:** Travel straight to the chosen sheep and push it directly at the gate. When the sheep is delivered, return back to Phase 1a. Continue this cycle until all black sheep are

delivered (the computed set of undelivered sheep is empty). If no white sheep made it to the other side, the simulator will stop automatically. Otherwise the player switches to Phase 2.

- **Phase 2:** Move the dog towards the gate. When the dog reaches the gate, switch to Phase 3a.
- **Phase 3a:** Choose a random, undelivered, unpursued ***white*** sheep and switch to Phase 1b.
- **Phase 3b:** Travel straight to the chosen sheep and push it directly at the gate. When the sheep is delivered, return back to Phase 3a. Continue this cycle until all white sheep are delivered (the computed set of undelivered sheep is empty). If no black sheep made it to the other side, the simulator will stop automatically. Otherwise the player returns to Phase 0.

Phases 2 and 3 are identical to Phases 0 and 1, with the distinction that Phases 0 and 1 tackle moving *black* sheep from right to left, while Phases 2 and 3 tackle moving *white* sheep from left to right. Given enough time, this strategy is sufficient to completely separate the two colors of sheep. In Phases 0 and 1, a *black* sheep is marked as undelivered if its x-coordinate is greater than 50, and in Phases 2 and 3, a *white* sheep is marked as undelivered if its x-coordinate is less than 50.

2.2 CONVEX HULL TRAVERSAL

For the basic mode, if the sheep-to-dog ratio is relatively low, our player will use the one-by-one strategy as explained above. If the ratio is sufficiently high (how this ratio is determined is discussed in Section 3.2), then our player will switch to a mass herding strategy. This strategy involves computing the convex hull of the set of sheep coordinates plus two points on the center fence and traversing the hull to push all of the sheep towards the gate.

The ***convex hull*** of a set of points is the smallest possible convex polygon such that all points in the set lie on or inside the polygon. Having a dog compute this hull for the sheep points and traverse it will allow it to exert influence on all of the sheep while being guaranteed to cover the entire set of sheep without pushing any sheep away from the gate. Essentially, the dog will compute the hull, traverse it to push the sheep in towards the gate, then recompute the hull with the new sheep positions and repeat. Our player uses Graham's algorithm to compute the convex hull. The algorithm can be roughly described as follows:

1. From the set of N points, find the rightmost lowest point (in the case of the simulator, this will be the point with the largest x-coordinate among the point(s) with the largest y-coordinate) and mark it $\boldsymbol{p_0}$.
2. Sort all other points angularly around $\boldsymbol{p_0}$ in ascending order. If two points form the same angle with $\boldsymbol{p_0}$, the one with the lesser distance wins out.
3. Push $\boldsymbol{p_{N-1}}$ and $\boldsymbol{p_0}$ onto a stack (in that order). These points are guaranteed to be on the hull.
4. Set $i = 1$. While $i < N$: If the point $\boldsymbol{p_i}$ forms a counterclockwise turn with the top two points on the stack, push it to the stack and increment i. If not, pop the top point from the stack, as it cannot possibly be on the hull. At the end of the loop, the stack will contain the hull points.

Figure 2a - 2d below visually depicts the convex hull computation process for a set of points.

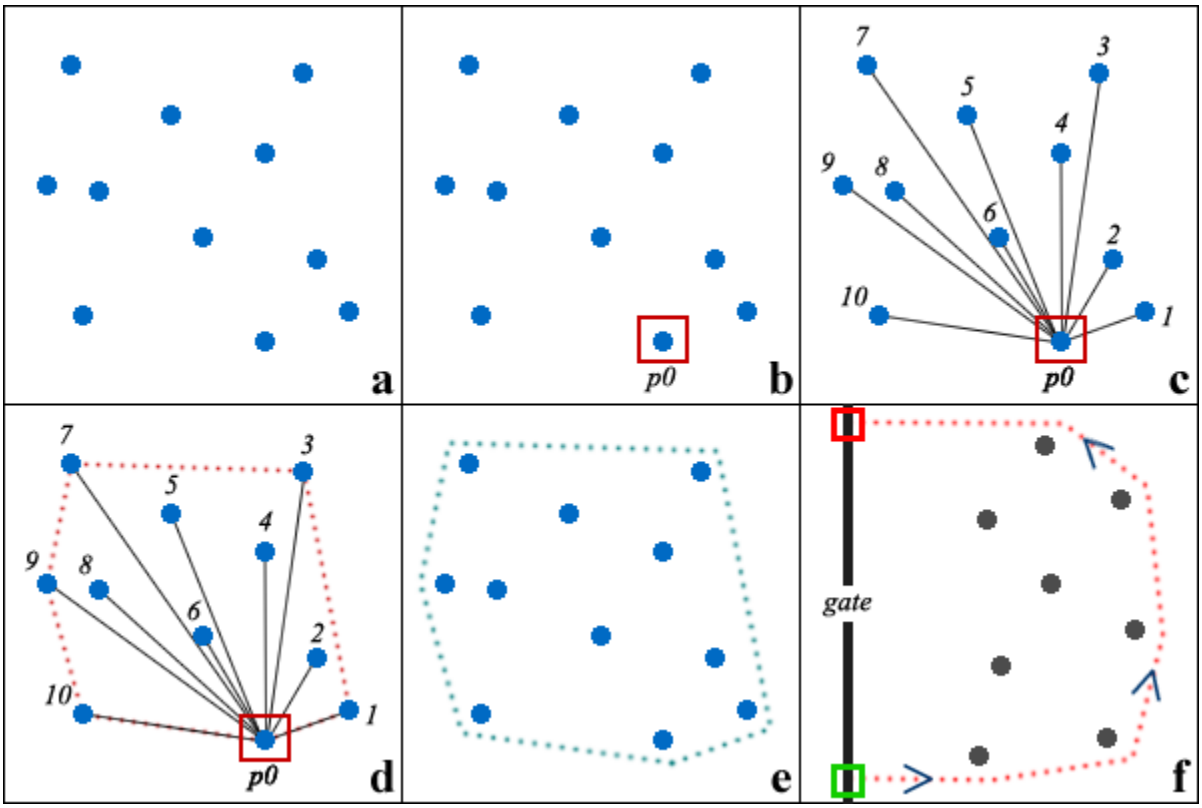

Figure 2: Demonstration of convex hull computation (a - d) and the result of computing the grown hull for traversal purposes (e). Figure 2f shows how these computations apply to our player on the field.

Our complete strategy using the convex hull computation is a three-phase process, outlined below:

- **Phase 0:** Assign the dog an initial movement delay based on its starting position and the number of dogs. When using multiple dogs, this will keep them relatively evenly spaced when traversing the hull to try to maximize the number of sheep influenced on each tick. Additionally, assign the dog a starting direction (clockwise or counterclockwise) based on ID. If the ID is even, start clockwise, otherwise start counterclockwise.
- **Phase 1:** Move the dog towards the gate. When the dog reaches the gate, switch to Phase 2a.
- **Phase 2a:** Use Graham's algorithm to compute the convex hull of the coordinates of each undelivered sheep plus two points on the fence located at the maximum and minimum y-coordinates determined by the group of sheep. Then grow out the hull shape to ensure that the dog moves *around* the sheep when traversing the hull. Switch to Phase 2b.
- **Phase 2b:** Have the dog traverse the grown hull, in the direction assigned in Phase 0, stopping briefly at each vertex to push sheep in the direction of the gate. When the hull has been traversed (all points in the hull have been visited), toggle the direction of the dog and return to Phase 2a.

For Phase 0, initial delay was assigned to a dog as follows: if there are only one or two dogs, there is no delay assigned to any dog. Otherwise, the initial movement delay for a dog is determined by the following estimation:

$$delay = 2p\left(\frac{T_{MAX}}{d}\right)$$

Where p is number of dogs away the player is from the center dog, T_{MAX} is the maximum wait time that any dog will have, and d is the total number of dogs. This formula will uniformly distribute delays among the dogs, with dogs at the center having no delay, and dogs at the ends having the most delay.

The value of p can be computed as follows: If there are an odd number of dogs, p is the distance from the current dog's ID to the center dog's ID. If there are an even number of dogs (i.e. two center dogs), p is the distance from the current dog's ID to the ID of the closest center dog. In the case of our player, T_{MAX} is a constant, set at 60 ticks. The justification for this value is explained in Section 3 of the report.

Phase 1 is once again quite simple. Move the dog to the gate at the center of the field and switch to the next phase when the dog arrives at the gate.

For Phase 2a, we compute the hull using Graham's algorithm as described above. It's important to note that simply computing the hull around all of the sheep and traversing it will only serve to collect the sheep at the center of their side of the field, and will not direct them to the gate. To ensure that the sheep get directed towards the gate, we introduce two additional points into our hull computation. We find the maximum and minimum y-coordinates from among the group of sheep and add the fencepoints $(50, y_{max})$ and $(50, y_{min})$ to the coordinate set. This will ensure that the dogs will only traverse the hull from fencepoint to fencepoint and will not push any sheep to the right, only left and towards the gate.

Figure 2f displays visually the new hull with the inclusion of these two points. This strategy could be problematic, however, in a case where all of the sheep are north (or south) of the gate, at which point the dogs will push the sheep towards some arbitrary fence location instead of the gate. To remedy this, we set a minimum vertical distance from the gate that the two fencepoints can be.

Through empirical testing we found a distance of about 8m to be satisfactory. It is close enough to the gate that dogs at those points will exert influence on sheep (recall the 10m radius of influence for each sheep) and far away enough that the sheep on the appropriate side of the gate won't be influenced vertically by dogs on the other side of the gate. What this ultimately means is that if y_{max} or y_{min} are less than 8m away from $y_{fence} = 50$, their values will be clamped to 58 and 42, respectively. Figure 3 below (left) shows a situation in which two dogs are positioned at these locations.

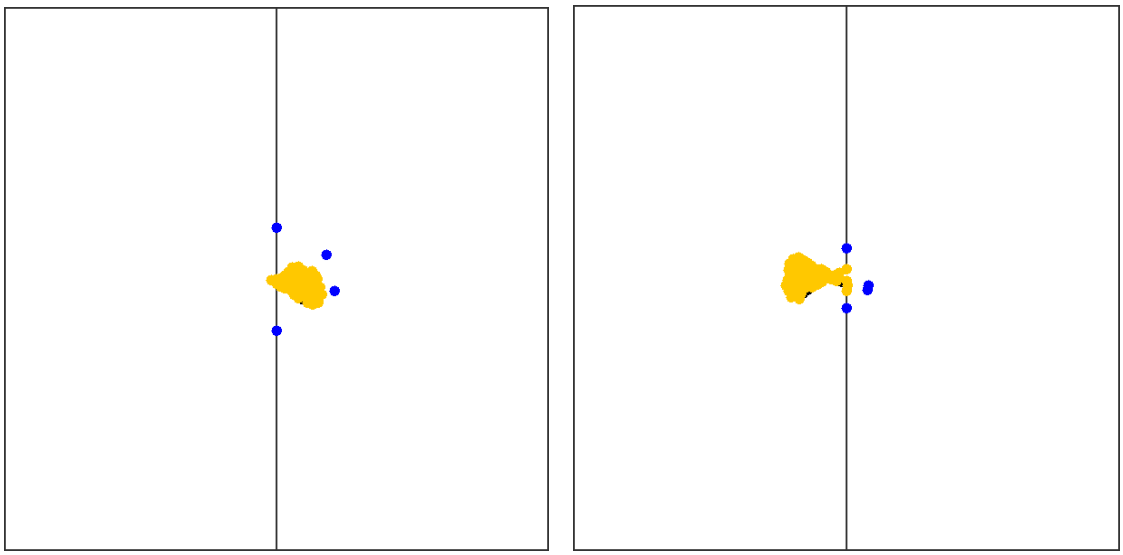

Figure 3: Two dogs at vertical fencepoints 8 meters away from the gate (left) and two dogs during endgame located 4 meters from the gate (right).

Notice that the hull in Figure 2f is slightly larger than the actual hull containing the sheep would be. This is because traveling the normal hull path exactly will not benefit us as we would be chasing the sheep that located at each vertex. Instead, we wish for our dogs to travel around the outside of this shape so that the sheep will be pushed inwards and cluster together. To do that we need to grow the hull uniformly outwards. Figure 2e demonstrates the result of growing the hull computed in Figure 2d. We achieve this growth by specifying a radius, adding five new points uniformly around each vertex of the hull with the given radius (north, south, east, northeast, southeast), and recomputing the hull with this increased set of points. We don't add any new points to the left of any vertex because we are not concerned with growing the shape leftward.

After Phase 2a, we have the travel trajectory we want the dog to travel. In Phase 2b, the dog traverses this travel shape, stopping at each vertex. From our earlier implementations, we observed that as our dog traversed the path, it had the least effect on the sheep that were close to the vertex of the hull. This was because at the vertices of the hull our dog alters its bearing, thus sometimes causing the sheep to be repelled back and forth, especially if the direction switch is sharp. As a result, the sheep closer to the vertices didn't cluster well with the rest of the sheep and delayed our performance. We decided to remedy this by making the dog pause at each vertex and deliberately making a one-tick push inwards (towards the gate), thus having a pronounced effect on the sheep near that vertex. From observing numerous runs, we determined that an inwards push of 0.9 meters works best. A smaller push didn't have enough of an effect on the sheep and we did not get the desired clustering; whereas, a greater push sometimes lead to the dog jumping over the closest sheep, thus causing the sheep to dart outwards, away from the gate. To ensure we don't push any sheep far left enough that they bounce off the fence and go the opposite way, we introduce a minimum x-value for pushing, set at 52. This way, no dog in this vertex pushing phase will push to a point with an x-value less than 52 meters. After completing this inward push, the dog backtracks and resumes its travel path around the hull. Once the hull has been completely traversed, the dog returns to Phase 2a.

2.3 ENDGAME STRATEGY

For the advanced task and one-by-one movement in the basic task, our player does not incorporate a distinct endgame strategy. For convex hull traversal, our player does make a few parameter changes when the task is approaching completion to assure that the sheep remain in the intended path. Our player determines that is has entered an *endgame state* if the maximum x-coordinate of any sheep is less than 51 meters, meaning all sheep are presumably within 1 meter of the fence, at least horizontally.

First, we discovered that as the group of dogs gets closer to the sheep, the sheep tend to start running. However, if the dogs continue to travel at maximum speed, they tended to cause the sheep to run back and forth in a similar manner to the dogs, instead of moving on a straight path to the gate. To remedy this, we decrease the dog speed in endgame to a leisurely 0.8 m/tick. Through testing we discovered that this speed allows sheep who are moving towards the gate to continue on their trajectory long enough to have the greatest chance of being delivered before their velocities reverse.

In addition to altering speed, we also alter the minimum vertical gate distance. As the x-coordinates of the sheep get closer and closer to 50m, the dogs need to make sure they aren't lining up and spreading out along the fence. To account for this, during endgame we cut the minimum vertical gate distance in half in an effort to corral the sheep faster. The effect of this can be observed in Figure 3, on the right.

Finally, our endgame player increases the growth radius of the hull. We noticed during testing that when the sheep become heavily clustered, the repulsion factor between sheep becomes more pronounced, and a number of sheep are actually able to escape the hull during endgame, causing a great deal of slowdown for our player. To counteract this, we scale the growth radius by a multiplication factor (detailed in Section 3) to continue to ensure that the path of the dogs will fully enclose the cluster of sheep.

3 JUSTIFICATION FOR STRATEGY

3.1 MAXIMUM INITIAL DELAY IN BASIC STRATEGY

Initially it would seem as though putting an initial delay on any of the sheep would cause an increase in task completion time. However, we found that attempting to evenly space out the dogs during hull traversal improved our numbers by a fair bit (several hundred ticks on average), and any increase in simulator time incurred from delaying dogs to achieve this spacing was

overpowered by the amount of time saved during hull traversal. The value of T_{MAX} as described in Section 2 was decided as follows:

The time it takes for a dog to traverse half of one side of a field at max speed can be computed as distance divided by speed, where distance is 50m (left) + 50m (top) + 50m (right) = 150m, and speed is 2m/tick. Thus, the resulting time is 150 / 2 = 75 ticks. To account for the time offset for different dogs to arrive at the gate, we then subtract the difference in time it takes to move from the far fence to the gate for a dog on the outside of the initial formation vs. a dog in the center of the formation, which, from observation in the simulator, is about 10 ticks. Finally, because we know that the initial hull will likely have a smaller (but not much smaller) perimeter than the field itself, we also subtract an additional 5 ticks to account for this smaller travel distance, leaving us with an estimated maximum delay of 60 ticks.

3.2 KEY RATIO FOR DETERMINING STRATEGY IN BASIC MODE

Within the basic mode, either the one-by-one or the convex hull strategy is used depending on the simulator's arguments. To determine when to use which strategy, we tested each strategy on different configurations. Contrary to our expectations, the optimality of a given strategy within each configuration depended not only on the ratio of sheep to dogs, but also the absolute number of sheep. The data indicated that for each absolute number of sheep, there is a key ratio of sheep to dog above which the convex hull strategy performs better and below which the one-to-one strategy performs better. This is because the convex hull strategy is a herding strategy which is able to more efficiently handle a large number of sheep with few dogs, whereas the one-to-one strategy is most efficient when each dog has a small number of sheep to chase.

For several values of number of sheep, we found this ratio by testing both strategies on several values of number of dogs. We then plotted this ratio of sheep to dog against absolute number of sheep and employed a variety of regression techniques such as linear regression, polynomial regression, power regression, and logarithmic regression. The best fit regression was a power ratio with an R^2 value of .968. This regression is shown in the chart in Figure 4.

As the chart and regression indicate, the ratio of sheep to dog above which the convex hull strategy is more efficient is not linear with number of sheep. The key ratio is approximated by the formula $330.26 \times (\#\ of\ sheep)^{-0.654}$. The high R^2 value indicates that this model is fairly accurate.

The data also showed that regardless of the absolute number of sheep, the one-to-one strategy is much more efficient than the convex hull strategy when there is only one dog available. This effect is so strong that one dog is probably a qualitatively different case which deviates from the usual pattern.

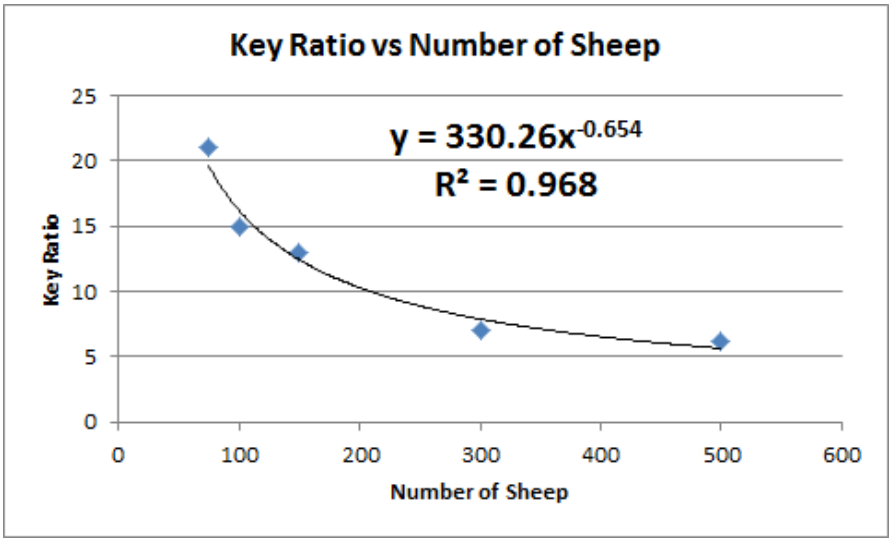

Figure 4: The relationship between the number of sheep and the key sheep/dog ratio needed to switch strategies.

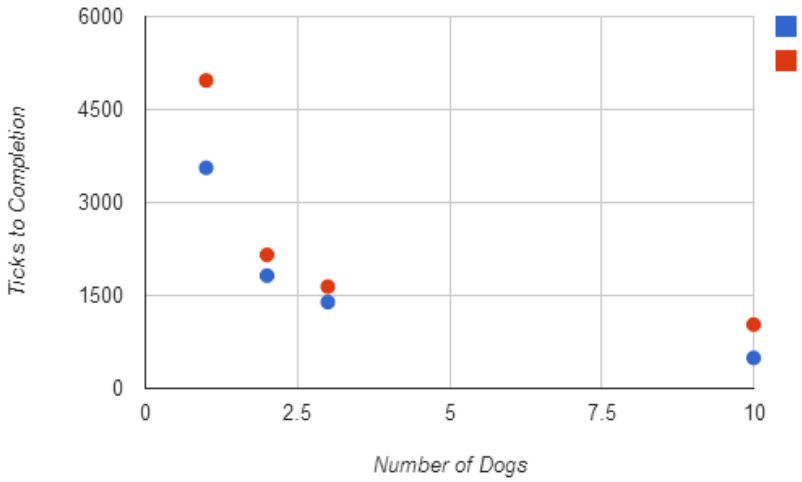

Figure 5: The relationship between the number of dogs and the average number of ticks to completion. The red points indicate trials where the convex hull strategy is used, and blue dots indicate the one-by-one strategy.

In Figure 5 above, ticks to completion for each strategy in basic mode with 20 sheep is shown. The red dots represent the convex hull strategy and the blue dots represent the one-to-one strategy. It can be seen that the discrepancy between the two strategies is much greater when only one dog is used. Because of this, whenever the basic game is run with only one dog, the one-to-one strategy is used regardless of the key ratio.

When the simulation starts, our AI computes the key ratio according to this formula. It then computes the actual ratio by dividing the number of sheep by number of dogs. If the actual ratio is greater than or equal to the key ratio and more than one dog is available, the convex hull strategy is

used. If the actual ratio is less than the key ratio or only one dog is available, the one-to-one strategy is used.

3.3 STATIC VS. DYNAMIC

In basic mode, we considered two ways to utilize the key ratio formula. The first is to compute which strategy, convex hull or one-to-one, is more efficient when the simulation starts and stick with that strategy throughout the duration of the simulation. This is a static strategy. The second way to utilize the ratio formula is to compute the ratio of undelivered sheep to dogs at each tick and compare it to the key ratio to determine which strategy to use at which tick. This is a dynamic strategy.

The dynamic strategy we tested computed the key ratio at the beginning of the simulation and compared the key ratio with the ratio of undelivered sheep to dogs. The ratio of undelivered sheeps to dogs decreases as the simulation progresses. Because of this, if the ratio of undelivered sheeps to dogs is below the key ratio at the start of the simulation then the one-to-one strategy is employed throughout. However, if the ratio of undelivered sheep to dogs is greater than or equal to the key ratio at the beginning of the simulation then the AI would transition from the convex hull strategy to the one-to-one strategy when the ratio of undelivered sheep to dogs falls below the key ratio as sheep are herded to the left of the fence. In this scenario, the dogs would initially use a mass-herding strategy, then transition to a one-to-one strategy as the number of sheep left to herd falls.

Testing data from a variety of configurations indicated that the dynamic strategy performed better than the static strategy in only ⅓ of configurations. The dynamic strategy usually fared better when there was a large number of both dogs and sheep. However, the dynamic strategy never improved performance by more than 20 ticks. In contrast, in cases where the static strategy performed better the tick difference between the strategies was often more than 100 ticks. Because of this, we reasoned that the dynamic strategy either reduced performance or did not provide a significant increase in performance. We utilized the static strategy in our submission.

3.4 PARAMETER OPTIMIZATION

Our AI utilized a number of parameters for the basic mode. These include the speed at which the dogs traverse the hull before the end game phase, the speed at which the dogs traverse the hull during the end game phase, the distance the convex hull is grown by to compute each dog's path, the multiplier on this growth distance when the end game state is reached, the x-value which all sheep have to be past to enter the end game mode, and vertical distance from the gate used when computing the each dog's path for both before and during the end game mode.

We optimized these parameters by choosing initial intuitive values followed by testing. We chose initial values by using reasoned guesses. To optimize the parameters, testing was done on a

standard configuration of 100 sheep and 5 dogs. We manually applied a basic hill-climbing algorithm. We changed one parameter at a time between each test. When a single parameter was optimized, another parameter was changed. Optimizing parameters improved our performance on the 100 sheep and 5 dogs configuration by about 120 ticks. The final parameter values are given in the following table:

Sheep Follow Distance *Distance behind a sheep a dog should try to stay in 1-by-1 strategy, in meters*	1
Hull Growth Radius *Distance to grow out the hull by*	1.5
Hull Growth End Game Multiplier *Multiplier on growth distance when endgame state is reached*	2.5
Hull Traversal Speed *How quickly the dog should traverse the hull, in meters/tick*	2
Vertex Push Distance *How far the dog should push towards the center at each vertex of the hull, in meters*	0.9
Minimum Push X Value *The minimum x value that a dog pushing towards the center can push to*	52
End Game Sheep X Threshold *The maximum x value of all sheep that determines when to enter endgame state*	51
End Game Dog Speed *How quickly the dog should travel in endgame state, in meters/tick*	0.8
Minimum Vertical Gate Distance *Minimum vertical distance from the gate when computing the hull, in meters*	8
End Game Minimum Vertical Gate Distance *Minimum vertical distance from the gate when computing the hull during the endgame state*	4

4 TOURNAMENT RESULTS ANALYSIS

Each group's final player submissions participated in a tournament in which the simulator ran each player's code in both basic and advanced modes for a variety of dog/sheep configurations. For each configuration tested, 10 trials were run for each player, and the average scores over the 10 trials for all configurations are shown in graphs in the following sections. The configurations used are displayed in the table below. For each column, *d* is the number of dogs, *S* is the number of sheep,

and *B* is the number of black sheep. The first 25 configurations are in basic mode (thus *B* is irrelevant) and the last 21 configurations are in the advanced mode.

Config No.	***d***	***S***	**Config No.**	***d***	***S***	***B***
1	1	5	*26 (Adv 1)*	1	50	5
2	1	50	*27 (Adv 2)*	1	50	45
3	1	200	*28 (Adv 3)*	1	50	25
4	1	500	*29 (Adv 4)*	1	200	20
5	2	50	*30 (Adv 5)*	1	200	180
6	2	200	*31 (Adv 6)*	1	100	100
7	2	500	*32 (Adv 7)*	11	50	5
8	4	50	*33 (Adv 8)*	11	50	45
9	4	200	*34 (Adv 9)*	11	50	25
10	4	500	*35 (Adv 10)*	11	200	20
11	7	50	*36 (Adv 11)*	11	200	180
12	7	200	*37 (Adv 12)*	11	200	100
13	7	500	*38 (Adv 13)*	32	50	5
14	11	50	*39 (Adv 14)*	32	50	45
15	11	200	*40 (Adv 15)*	32	50	25
16	11	500	*41 (Adv 16)*	32	200	20
17	20	50	*42 (Adv 17)*	32	200	180
18	20	200	*43 (Adv 18)*	32	200	100
19	20	500	*44 (Adv 19)*	100	200	20
20	32	50	*45 (Adv 20)*	100	200	180
21	32	200	*46 (Adv 21)*	100	200	100
22	32	500				
23	100	50				
24	100	200				
25	100	500				

From the results data we received, we will assess our overall placement in the tournament, as well as provide a discussion on our own player's performance and how it is affected by these changes in configuration.

4.1 OBSERVATIONS OF OVERALL RESULTS

WHERE WE STAND

We separated the data for the basic and advanced tasks to see how different groups fared in the two distinct tasks. There were huge variations in score, especially in configurations where timeouts had

occurred. To better understand the overall performance we averaged the scores that the groups got for each configuration. We then normalized the average score obtained for a given configuration by the average score of all groups for that configuration and plotted the graphs. To better see the variations in performance the scores that were worse than 2.5 times the average are clamped to 2.5.

Figure 6 below summarizes the performance for basic task. It can be observed that our players' performance (in black) follows the general trend of scores but tends to perform slightly worse than most of the other players. However, we do better than the average of the average scores 80% of the time. The cases where we do worse than the average are generally those where the ratio of sheep to dogs is big enough for us to be using the herding strategy but is not extremely large.

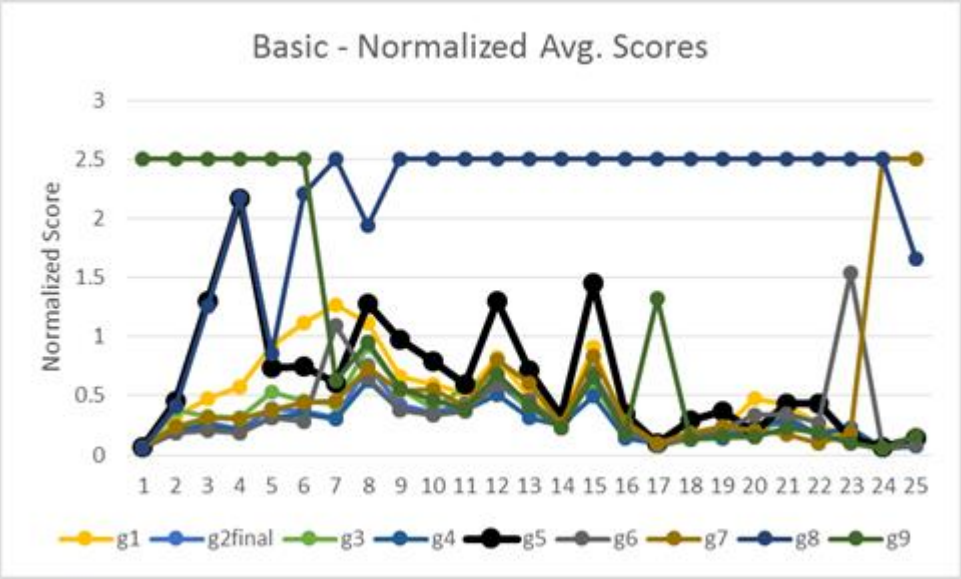

Figure 6: The average scores of each group for each basic task configuration normalized by the average scores of all groups.

As this ratio increases our relative performance improves (we are never below average for 500 sheep) because a herding strategy becomes more effective as the sheep-to-dog ratio increases. Our performance is also significantly off when we are only operating with one dog (and employing a one-to-one strategy) and the number of sheep are in the hundreds (configurations 3-5). This is because it becomes an immense workload for the lone dog to fetch each sheep one by one. However, we never time out and are never last.

Next, we look at the advanced task. We only employ a one-to-one strategy for advanced tasks and our relative performance is better here as compared to for basic task. The performances of each group are summarized in the graph above. It can be seen from Figure 7 below that we follow the trend more closely here. We are worse than the average only once and our rank for different configurations also improves. Once again, we are always able to finish the task and never time out.

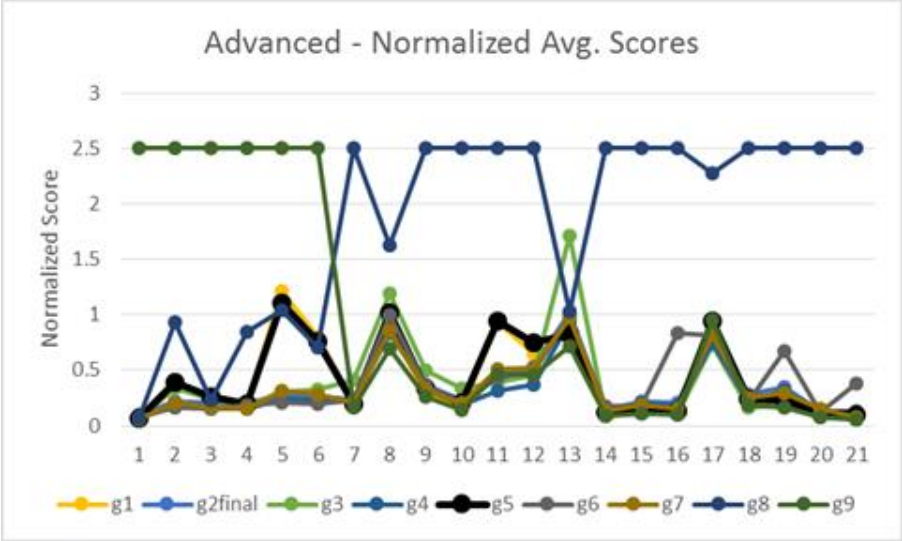

Figure 7: The average scores of each group for each advanced task configuration normalized by the average scores of all groups.

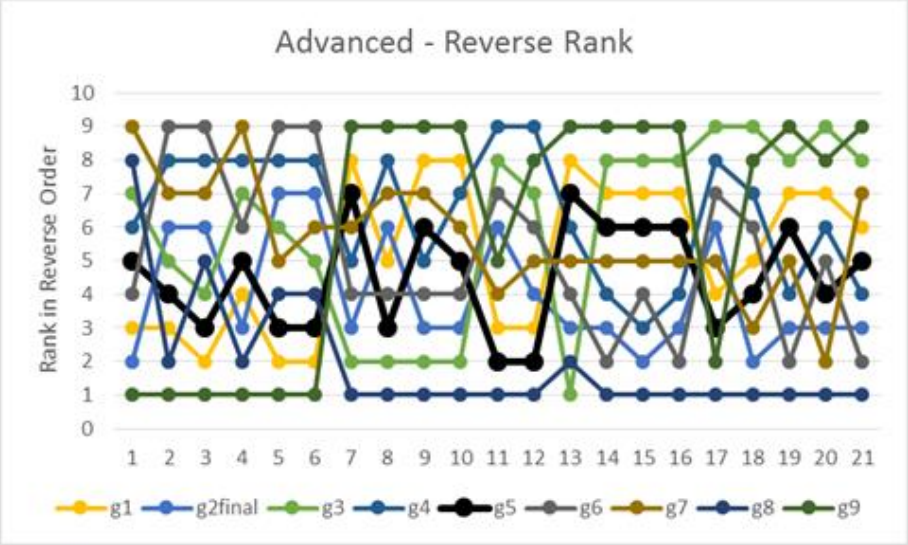

Figure 8: The rank (in reverse order) obtained by each group for each advanced task configuration.

Figure 8 above shows our rank in these advanced task configurations compared to the other players'. We are ranked 8th in cases where the black sheep are 50% or more of the total sheep (Advanced config. 11 & 12) but we do much better in other cases. This is because we always employ a one-to-one strategy for advanced tasks; a herding strategy for cases where black sheep are in higher percentages would have helped improve our relative performance.

4.2 ANALYSIS OF OUR PLAYER'S PERFORMANCE

We will now zero in on our own player's performance to try to address how it may be affected by different game configurations. Figure 9 displays, for each configuration, our performance as a

percentage of the best (i.e. lowest) score achieved by any player in that configuration. From this graph, a number of observations can be made.

Recall from the chart at the beginning of this section that the first 25 configurations were run in the basic mode. We can see from the graph that our player's scores were on average farther from the best score in most basic mode configurations.

The most significant factor in this trend may very well by that our clustering is inefficient. Our player attempts to cluster all sheep together instead of breaking up the work, and as a result it is difficult to make the sheep run (or move quickly at all) towards the gate. So while we were able to complete the task in every configuration, because the sheep do not move quickly with the hull strategy, the simulator's performance measure ranked us lower.

While the average performance in basic mode is lower than advanced, it is worth observing that there is a noticeable cyclic pattern in the first half of the graph. The lowest points represent configurations where the sheep/dog ratio is closer to 1, and thus our player uses the one-by-one strategy. However, as the number of sheep increase while keeping the number of dogs constant, our performance is diminished, causing the series of local maxima. This is because the one-by-one strategy scales linearly in time with the number of sheep, and, as mentioned previously, once the ratio is large enough to switch to the hull strategy, the average speed of the sheep is much slower.

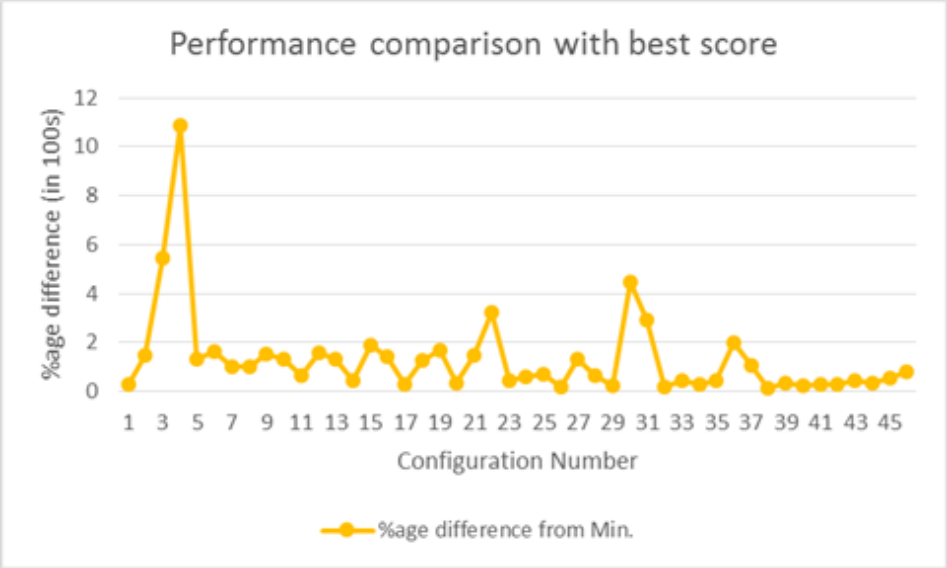

Figure 9: Our player's performance as a percentage of the best performance in each configuration.

The outlier (1100%) is the result from configuration 4, in which there is 1 dog and 500 sheep. Unfortunately our hull strategy is not guaranteed success with 1 sheep, and due to time constraints we were not able to come up with an efficient one-dog strategy, so our player will always use the one-by-one strategy when there is 1 dog.

The last 21 configurations were run in the advanced mode, in which the player only needed to herd across the black sheep. Because we use our one-by-one strategy across the board here, our average performance was closer to best since, as noted in class discussion, many groups utilized a one-by-one solution for advanced mode, at least for when the number of black sheep was sparse. This also explains why there are spikes in performance at configurations, 27, 30, 31, and 36.

Many players implemented a clustering strategy for when the number of black sheep becomes dense or reaches a percentage of at least 50% of the total sheep. These clustering strategies are likely more efficient at getting all of the black sheep across, and the amount of time it takes to return the white sheep is almost inconsequential. Because our player always uses one-by-one, we suffer the same fate we saw in basic configuration 3; when the number of sheep to gather is much greater than the number of dogs, our performance suffers noticeably.

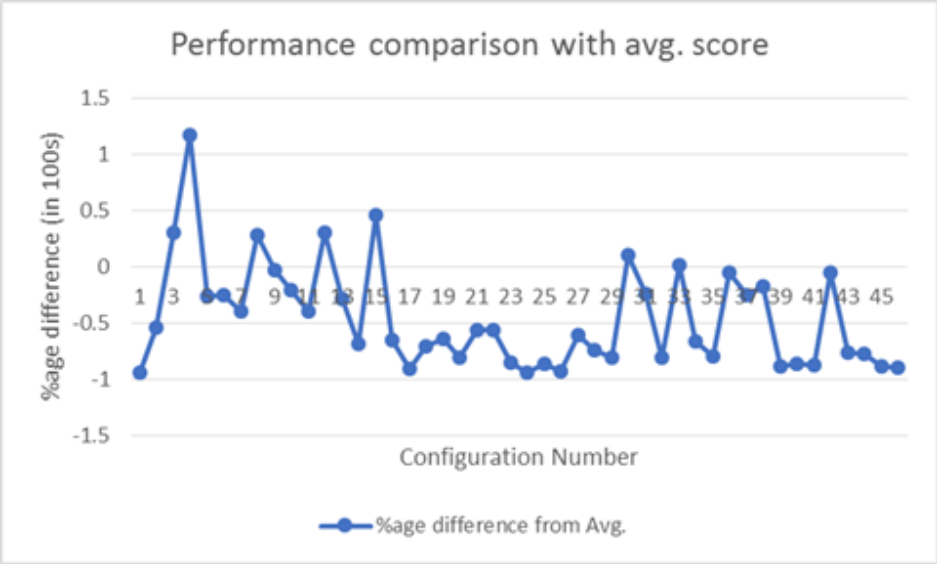

Figure 10: Our player's performance as a percentage of the average performance in each configuration.

Figure 10 summarizes our performance comparison with the average score of all players. As briefly mentioned in Section 4.1 our scores are consistently better than the average scores except for a few exceptions. We are worse than the average only 13% of the time. Only one of these instances is for the advanced task and it is configuration 30. This is when we have one dog at our disposal and we need to move 180 sheep across the fence and our one-to-one strategy makes it difficult for our program to do the task efficiently. A better strategy could've been employed that performs some kind of herding. The other four times we did worse was in basic task configurations and the reason for that has been addressed in section 4.1.

There are several instances where we do much better than the average (50 - 100% better). These are mostly cases where we have a large number of dogs at our disposal (configuration 17-26). In such scenarios our dogs are spread around the convex hull and are always pressing the sheep inwards from all sides or they are in one-to-one mode and are distributing work effectively.

SIMULATOR TIMEOUTS

By ensuring that we had covered all possible scenarios and by employing strategies that scaled reasonably well with number of dogs and sheep, we were able to develop a player that completed all runs successfully and never times out.

5 FUTURE IMPROVEMENTS

Our AI can undergo a number of improvements. When testing the dynamic strategy described in Section 3.3, we computed the key ratio based on total number of sheep in the game and the actual ratio based on the number of undelivered sheep. However, the dynamic strategy could be improved by dynamically calculating not just the actual ratio of undelivered sheep to dogs, but also the key ratio. Instead of basing the key ratio on total number of sheep to dogs, it could be based on number of undelivered sheep to dogs. In this way, both the key ratio and the actual ratio would be dynamically adjusted to provide a better metric of when to use the convex hull strategy and when to use the one-to-one strategy.

Another potential improvement is to determine a way to more quickly bring in the corners of the convex hull. When using the convex hull strategy, the sheep often form a shape in which two corners on the right side stick out as demonstrated in Figure 11 below. This causes the dogs to be a fair distance from the many of the sheep. Because of this, the sheep are not herded towards the fence opening as fast as they could be if the dogs were closer to them. By pushing in the corners of the convex hull further, the efficiency of the convex hull strategy could be improved by ensuring the dogs are close to as many sheep as possible. This could be implemented by having the dogs spend extra time pushing the sheep towards the center when the dogs are at corner vertices.

Additionally, our AI could be improved by implementing combinatorial testing during parameter optimization. Many of the parameters mentioned in Section 3.4 such as Hull Growth Radius and Hull Growth End Game Multiplier are interlinked. Changing one variable at a time optimizes each variable given the values of other variables. This may miss groups of optimal values. Because of this, combinatorial testing may provide better optimization than changing one variable at a time.

Furthermore, our AI could be improved by using more than two phases in the convex hull strategy. Currently we divide the convex hull strategy into end game and pre-endgame. Factors such as the speed of the dogs and minimum vertical distance from the gate differ in the two phases, and the phase transition occurs when the maximum sheep x-location falls below a certain value. Three or more phases could be implemented through a similar transition mechanism with varying dog speeds, hull growth radius multipliers, and minimum vertical distances to the gate. This would allow optimal dog movements, depending on how close to the gate the group of sheep is. Parameters could then be optimized through testing.

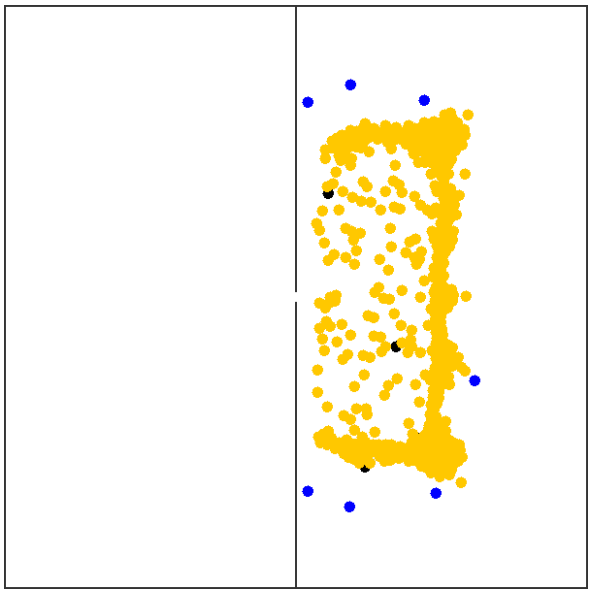

Figure 11: Concentration of sheep at the corners of the hull. This tended to cause an overall slowdown in the hull traversal process.

Finally, our AI could be improved by implementing a clustering strategy for the advanced game mode. The convex hull strategy intuitively would not provide an effective clustering strategy for the advanced game mode because it would herd many more white sheep than necessary to the left side along with the black sheep. A tree structure clustering strategy such as that implemented by Group 4 would be much more efficient because it would push individual black sheep rather than ensnaring all sheep.

6 CONCLUSION

In the tournament, our AI performed much better on a relative basis in the advanced mode instead of the basic mode. In the advanced mode, we used a relatively less complex strategy. Instead of a convex hull, we only used one-for one to sort the sheep onto the correct side of the fence by color. Despite being much simpler, the strategy led to a better tournament rank. Clearly, complexity of strategy and performance are not necessarily correlated.

Although the average performance of our basic strategy was not as efficient as the strategies of other players, it is worth noting that our strategy is scalable to a very large degree. That is to say, some groups seemed to have taken the geometry of the field to be constant and devised some of their strategies to take advantage of the specific dimensions of the field. An advantage that our strategy has is that it will operate successfully independently of the size and shape of the field, and does not require the implementation of any additional sub-strategies to successfully capture all of

the sheep, assuming the field is also of a convex shape. Our strategy as it stands would not actually be successful in the case where the field itself is not convex, but a small modification could be made to detect where the hull intersects the field boundaries and adjusts the shape to account for it.

The sheep herding AI has many applications. Beyond the obvious sheep herding, the advanced AI strategy can be used whenever one specific needs to be moved across a membrane. This situation arises often in cell biology where various molecules have to be segregated across the cell membrane in order to maintain homeostasis. Synthetic biology in particular has use for sheep herding algorithms developed in this project as artificial intelligence becomes integrated with biological engineering. Perhaps in the future nanorobots will be able to perform the function of the dogs in the simulation, "herding" materials across cell membranes to maintain homeostasis.

7 APPENDIX

7.1 INDIVIDUAL CONTRIBUTIONS

SAM
For the player, contributed code for the advanced strategy, ran empirical trials to compute the formula for the key ratio, and did some parameter optimization. For the report, contributed to strategy justification, tournament results, future improvements, and the conclusion.

ANDREW
For the player, contributed a large amount of code for both strategies, spearheading the convex hull strategy in the basic mode, as well as the final advanced mode strategy. For the report, wrote the problem statement, strategy section and contributed to tournament analysis.

MAHD
For the player, contributed code for phase detection and traversing the convex hull, and created many convenience functions used by the player, such as the *moveDog* function. For the report, crunched the tournament data and contributed to the tournament and strategy sections.

7.2 ACKNOWLEDGEMENTS

We would like to thank Jiacheng for implementing and improving the simulator throughout the duration of the project and for running the official tournament. We would also like to thank Professor Ross for creating this complex and interesting project and for inciting thought-provoking class discussion. We thank the other groups for improving their players throughout the project so that we could learn from their successes and failures and test their players against our own.