Analysis: Sheepwalking

Herding strategy

What is the most efficient way to herd Bleatrix from cell (**X**, **Y**) back to the "origin" cell (0, 0)? We will simplify our discussion by assuming that **X** and **Y** are both nonnegative; because Bleatrix's two-dimensional field is symmetric across both axes, our solutions can safely work with the absolute values of **X** and **Y**.

If Bleatrix is ever at a cell (x, y) such that $x \neq 0$, and $y \neq 0$ — that is, she is not along one of the two "axes" of cells — then two of her possible moves will take her toward the origin cell (reducing either her horizontal or vertical distance from it), and the other two will take her away from the origin. In this case, we should use the two sheepdogs to block the two "away" moves. Any move away from the goal in some direction would need to be reversed later on, costing us two moves. Moreover, we do not want to spend both sheepdogs to block opposite directions of movement (e.g., placing them to Bleatrix's left and right to force her to move either up or down), because then she would make a purely random horizontal or vertical walk, with no tendency toward the goal.

The only remaining case — apart from being at the origin and thus being done — is that Bleatrix is at a cell of the form (x, 0) or (0, y). That is, she is along one of the two main "axes" of unit cells; without loss of generality, we will suppose she is at a cell of the form (0, y).

We know from the argument above that we do not want to use both sheepdogs to force her to randomly walk along the y-axis, so we should use one sheepdog to block her movement along the y-axis away from the origin. But should we deploy the other sheepdog to block one of her moves perpendicular to the axis? If we do not, she will move toward the goal with probability ⅓ and perpendicular to it with probability ⅔; if we do, she will move toward or perpendicular to the goal with equal probability. Since moving toward the goal is better (again, any lateral move needs to be undone eventually), we should use the additional sheepdog. For convenience, we will always deploy it in a way that keeps both of her coordinates nonnegative — that is, below her if she is along the x-axis, and to the left of her if she is on the y-axis.

A more rigorous proof of the optimality of our strategy follows at the end of this analysis.

Simulation can only get us so far!

Now that we have a strategy, a natural approach is to simulate it and take the average of, say, a million runs. Test set 1 only includes nine distinct cases; the others are symmetric, differing only in signs and/or in which values are X and Y. How hard can it be to get those nine answers?

As it turns out: pretty hard! The problem is that the length of our random walk has very high variance, so it is hard to get a confident estimate of the true expected length. A straightforward simulation solution will either take too long or not meet the strict tolerance requirements of the problem. However, we can run simulations offline and inspect the results, and we might notice that the answers are always close to a value of the form A / 9, where A is some integer. This pattern will not hold up for X and Y values outside of the [-3, 3] range, but it can serve us well here — an offline simulation can get us close enough to the true answers that we can confidently guess the value of A for each case, and then submit a solution that packages up those answers.

It is also possible to solve Test set 1 by hand, using the same method that also leads to a solution for Test set 2...

Expected herding time

Notice that the problem is "memoryless"; if Bleatrix moves to cell (x, y), the expected number of additional moves to reach the origin from there is the same as if she had begun at (x, y). So, to calculate the expected number of moves needed from a given starting point, we can use a series of *recurrences*. Let T(x, y) be the expected number of moves when starting from cell (x, y); trivially, T(0, 0) = 0. Let us consider our cases from above:

- Some cell (x, y) not on an axis: Bleatrix will move either left or down with equal probability; either way, this uses one move. So we have T(x, y) = ½ T(x-1, y) + ½ T(x, y-1) + 1.
- Some cell (0, y) on the y-axis: Bleatrix will move either down or right with equal probability; either way, this uses one move. So we have T(0, y) = ½ T(0, y-1) + ½ T(1, y) + 1. (The expression for a cell on the x-axis is similar.)

We can simplify the second recurrence by replacing T(1, y) with its representation from the first recurrence: ½ T(0, y) + ½ T(1, y-1) + 1. Then, after some algebra, the second recurrence simplifies to T(0, y) = ⅔ T(0, y-1) + ⅓ T(1, y-1) + 2. Now we are expressing T(0, y) only in terms of recurrences for smaller values of y. So, all told, for any starting cell, we have T(x, y)=

- 0 if x = y = 0.
- ⅔ T(0, y-1) + ⅓ T(1, y-1) + 2, if x = 0 but y ≠ 0.
- ⅔ T(x-1, 0) + ⅓ T(x-1, 1) + 2, if y = 0 but x ≠ 0.
- ½ T(x-1, y) + ½ T(x, y-1) + 1, if x ≠ 0 and y ≠ 0.

At this point, we can use recursion plus memoization — or some other form of dynamic programming — to quickly compute any answer we want. We will never need to evaluate T() for any particular cell more than once, so we have tamed the infiniteness and randomness inherent in the problem.

If we use, e.g., C++ `double`s, might our solution fail due to accumulated floating-point errors? This phenomenon is often worth worrying about, but in this problem, it is not even close to posing a threat. The total number of calculations needed is not very large — only a few per recurrence. Even if our solution for e.g. the case `500 500` involves calculating the answer for every possible distinct test case along the way, we will have at most a few million chances to introduce a tiny amount of error. A standard double precision floating point number has 15 decimal digits of precision, but we only need the first 6 to be correct, so even if our errors pathologically did not cancel each other out at all, we would be fine.

A (perhaps) surprising property of the solution

Check out the answers for the cases `100 0`, `100 1`, ..., all the way up to `100 30`. They are all (to nine decimal places) the same: 201.500000000. The answer for `100 100` is 212.727275769. Why are these values so close? That is, if we compare starting 200 moves away from the goal at (100, 100) versus starting only 100 moves away from the goal at (100, 0), why does the latter not help us very much?

Observe that, in our strategy, once Bleatrix has made enough left moves to reach the y-axis, her horizontal moves will alternate between right and left; once she has made enough down moves to reach the x-axis, her vertical moves will alternate between up and down. But regardless of where we are in the strategy, she has an equal probability of moving horizontally or vertically. Given that Bleatrix starts at (**X**, **Y**), we can be sure that she will reach the goal only after she has made at least **X** horizontal moves *and* at least **Y** vertical moves.

How many moves will it take until the first time at which both of those conditions are satisfied? Without loss of generality, let us assume that **X** ≥ **Y**. If **X** is much larger than **Y**, then by the time we get our **X**-th horizontal move, we will almost certainly have gotten **Y** vertical moves, so we can ignore **Y**. Then the expected number of moves needed to get **X** horizontal moves is 2**X**, since the probability of a horizontal move is ½. However, if **X** is not much larger than **Y**, or equal to **Y**, we cannot ignore **Y**, and it may take more than 2**X** moves in expectation for *both* conditions to be satisfied.

The above explains why the (100, 100) answer is larger than the (100, 0) answer. It also explains why it is not *too* much larger, since a large discrepancy would imply that it is very likely to get 100 moves in one direction *long* before getting 100 moves in the other direction.

However, why is the (100, 0) answer 201.5 rather than 2**X** = 200? Consider the first time at which we have at least **X** horizontal moves and at least **Y** vertical moves; call the number of moves M. We have just hit one of the axes for the first time, and we will be on the other axis (and thus done) if M matches the parity of **X** + **Y**, or one cell away otherwise. We know that T(0, 1) = T(1, 0) = 3; for large **X** + **Y**, we would expect parity effects to even out, so this final herding process should add ½(3) = 1.5 more moves to the expected total.

Also note that even though our chosen precision level obscures it, the (100, 0) answer is in fact slightly smaller than the (100, 1) answer, and so on.

Appendix: optimality proof

T(0, 0) = 0, and for any other (x_0, y_0), $T(x_0, y_0)$ is the minimum of 1/k × the sum of the T values of any k neighbors of (x_0, y_0), where k ≥ 2. Observe that k can always be 2 — it is never suboptimal to place 2 dogs adjacent to the current cell, since they block the neighbors with the larger values of T, and decrease our average T over all remaining neighbors. Therefore, to compute $T(x_0, y_0)$, we will pick any smallest two among $T(x_0 - 1, y_0)$, $T(x_0 + 1, y_0)$, $T(x_0, y_0 - 1)$, and $T(x_0, y_0 + 1)$, and take their average.

Lemma 1: $T(x_0 - 1, y_0) \le T(x_0 + 1, y_0)$ for all $x_0 > 0$, $y_0 \ge 0$.

Lemma 2: $T(x_0 - 1, y_0) \le T(x_0, y_0 + 1)$ for all $x_0 > 0$, $y_0 \ge 0$.

By Lemmas 1 and 2, $T(x_0 - 1, y_0)$ (and, by symmetry, $T(x_0, y_0 - 1)$) are better than the other two neighbors, so we place the sheepdogs to block the other two neighbors.

Proof of Lemma 1: Let *S* be an optimal strategy starting from $(x_0 + 1, y_0)$. From $(x_0 - 1, y_0)$, we could use a strategy *S*' that, as long as we don't get to the column of cells $x = x_0$, always places the dogs in a way that "mirrors" their placements in *S* with respect to the column $x = x_0$. Call this stage 1. When we first touch that column, we switch to using strategy *S*; call this stage 2.

During stage 1, if strategy *S* is in cell $(x_0 + a, b)$ after k moves, then *S*' is in cell $(x_0 - a, b)$. During stage 2, after k moves, they are always in the same cell. Since the path from $(x_0 + 1, y_0)$ to (0, 0) must cross column $x = x_0$, using this strategy means that for any random choices, either our modified strategy reaches (0, 0) before crossing column $x = x_0$, and therefore before strategy *S*... or the two strategies reach (0, 0) during stage 2, that is, both at the same time.

Sketch of proof of Lemma 2: This is similar to our proof of Lemma 1, but we mirror across the diagonal $y = x_0 + y_0 - x$. From $(x_0, y_0 + 1)$, the path must cross that diagonal, so we can define *S*' with similar stages, and a similar argument demonstrates the inequality.