Effect of Latency on Pursuit Problems

William Birmingham
Shane Rose
Gregory Miller
Matthew Mahan
Grove City College
Computer Science Department
Grove City, PA 16127
wpbirmingham@gcc.edu

Abstract

We model the pursuit problem as a set of distributed agents communicating over a network subject to latency. Latency has serious deleterious effects on solving the pursuit prob lem. In this paper, we present a simple, yet effective way of dealing with latency that yields very good performance. Our method disperses predators within a region in which the prey may move that accounts for network latency.

Introduction

We are interested in developing distributed AI systems (DAI) for networked, mobile gaming. In these games, the computation is distributed among the devices participating in the game. Each device has its own agent that maintains its local state, and there is no shared memory. When an agent needs to communicate state to another agent, such as updating its position, it sends a message over the Internet.

Because the messages are sent over a real network, there will be message latency. Consequently, the state that each device maintains is necessarily out-of-date with respect to all other devices, since it is receiving a message that is stale from the perspective of the sender. (Lamport 1978)

In the games that we design, agents must coordinate their behavior to achieve the game's objective. Many of the behaviors we design into our agents can be modeled, at least at a simple level, by the pursuit problem, as has been done by other game AI researchers. (Wittkamp, Barone and Hingston 2008) (Yannakakis and Hallam 2004) We must account for latency in the communication among agents. As we show in this paper, latency has a significantly negative effect on the performance many algorithms used to solve the pursuit problem.

Our approach treats all information we receive as stale. Using the simple ideas of movement radius and dispersion strategy, we direct pursing agents to an area where the prey must reside, rather than to a specific location. We show our algorithm is robust under even substantial network latency.

The Pursuit Problem and Solutions

There are many forms of the pursuit problem. Here, we give a general form for the problem and overview a few methods of solving it.

We define the pursuit problem as follows: set of prey $p_i \in \mathbf{P}$; and, a set of predators $r_j \in \mathbf{R}$. The objective of the problem is for one or more predators to capture the prey in a world while communicating over a network. Capture has different formulations: either the prey must be surrounded by predators, or a predator(s) must occupy the same space as the prey.

The predators and prey are characterized as follows: velocity, location, sight distance, kill distance, size of the agent, movement algorithm and, frame time (*f*).

The movement algorithm determines how the predators and prey determine their next move. There many algorithms for movement that range from pure chase and evasion (Korf 1992), to learning (Denzinger and Fuchs 1995), to those emphasizing "teamwork" in the context of a game (Heckel and Youngblood 2010) (Brown, et al. 2005) (Isaza, et al. 2008). These approaches, however, do not generally consider delays.

Some researchers have considered the use of particle filters for estimating positions of characters for a variety of different game types. (Bererton 2004) (Southey, Loh and Wilkinson 2007) (Klaas, Southey and Cheung 2005) (Weber, Mateas and Jhala 2011) (Hladsky and Bulitko 2008) These approaches do not explicitly consider latency in determining movement.

Frame time is an important parameter that describes how much real time is allowed for the predator or prey to determine its next move. This time includes all the time for computation as well as reading and writing the network.

When considered at all in a game, network latency is treated outside the movement algorithm, where "smoothing" is used to hide latency.

Generally, *f* is a lower bound on frame time. Most console games, for example, try to maintain a sixty frame per second frame rate, or 16.7 ms per frame. Thus, all computation and network communication must be done in under 16.7 ms to sustain frame rates. While this is not a hard bound—slower frame rates are acceptable for brief periods—maintaining frame rate is a significant concern when designing an algorithm.

Some algorithms assume that agents "know" something about the game state that other agents do not, such as the position of the prey. In these cases, the agent "sends" the position of the prey to the other agents. However, these data are not sent via a network connection with its concomitant latency but assume that once data are communicated, they are instantly and perfectly known to all other agents. (Korf 1992)

The network is characterized as follows: bandwidth allowed per agent*;* packet loss*;* packet latency, clock skew as seen by an individual agent*; and,* quality of service. The bandwidth allowed per agent is typically defined by the network SDK. In Microsoft's XNA, data is limited to 8K bytes per second.

Packet latency, a property of the network, can range into the 100s of milliseconds. (Claypool and Claypool 2006) At that delay, nearly six frames have passed (assuming f = 16.7 ms) by the time the network has simply transmitted a frame. In this paper, we do not study the effects of packet loss or clock skew, except to lump it into latency (e.g., reliable protocols, such as TCP, retransmitted lost packets).

The deadline implied by *f* together with latency makes it impossible for agents to fully describe their world models to all other agents. Since *f* is fixed, latency must be a primary consideration in designing algorithms.

The "world" can be characterized as follows: grid or non-grid, shape of the grid element (e.g., square, hex), size of the world, shape of the world (e.g., rectangle, octagon), presence of obstacles, and wrapped or non-wrapped. A wrapped world means that there are no boundaries at the end of the world; rather, one "edge" wraps to the corresponding edge on the other side of the world.

The Effect of Latency

For this paper, we cast the pursuit problem as follows. All agents play using a common world ("map"), which is non-gridded, obstacle-free, and wrapped. The predators are all of the same type, and there is a single prey. All agents operate on their own computer and communicate with each other only via network connections. The game ends when at least one predator is within the kill radius of the prey. The network quality of service is similar to UDP.

We allow the predators to coordinate by "announcing," i.e., sending a message over the network, the prey position to each other when the prey is seen. This positional information guides other predators to the prey for capture.

A condition of our setup is the presence of latency: every agent's model of the world—the locations of other predators and the prey—is stale. This is significant, since algorithms generally assume that the predators instantly know the prey's correct position once it is broadcast. (Korf 1992) However, stale data gives the prey a significant advantage: predators move to the position of where the prey *was*, not to the prey's current position.

In Figure 1**,** we show the experimental effects of latency on capture. In experiment that produced the data for the graph, there are three predators and a single prey. The y-axis shows the number of iterations it takes for a predator to capture the prey from random initial positions of both the predators and prey. The x-axis shows the latency in the network in milliseconds; there is no packet loss in this experiment. Each data point in the graph represents 100 runs of the pursuit game with the same initial conditions. The world is a grid of 10x10 units, where a unit is the size of a predator or prey. When a predator or prey gets to the "edge" of the world, it wraps. The prey moves at twice the predator's speed. Both predators and prey are able to see three "units" ahead, and the predators announce the position of the prey. An iteration consists of all the predators sensing, then moving followed by the prey sensing and

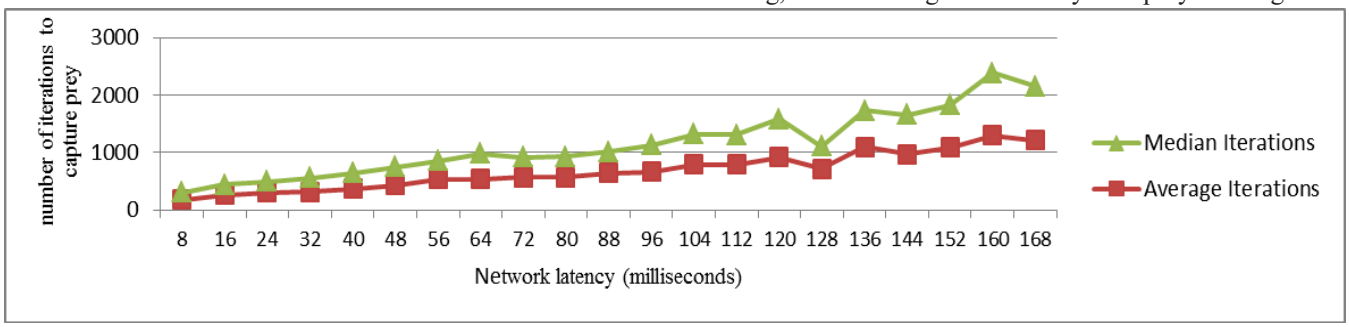

Figure 1: Effect of latency on capture rate.

then moving. The prey will move diametrically away from the closet predator, if the predator is within its sight distance of three units. Each iteration has $f = 16.7$ ms.

The movement radius is given by: $movRadius = (v_{prey} * p_{latency})$, where v_{prey} is the prey speed or velocity and $p_{latency}$ is the network latency.

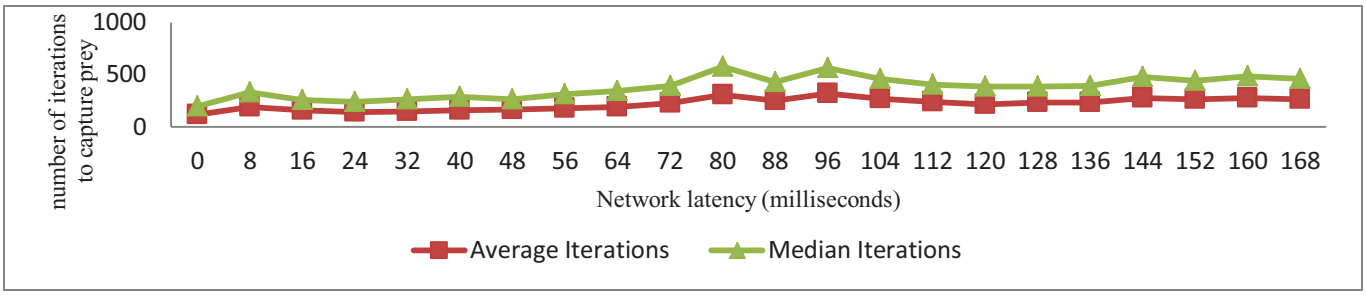

Figure 2: Effects of latency using target area "random" strategy.

The data in the graph show that the number of iterations needed to capture the prey increases as latency increases. The data show that the increase is nearly an order of magnitude in iterations, both average and median, as latency increases.

Accounting for Latency

The typical predator "announcement" algorithm has predators searching some space, through a variety of methods ranging from random movement to predetermined patterns. Once a predator has "seen" a prey, it announces the location for the other predators to converge upon. The announcement is a message with the observed location of the prey. However, both the location and the convergence methods have trouble in a network with latency.

The convergence methods, which vary among different approaches, generally direct the predators to the prey, usually with a provision to perform additional "looking' along the way as the prey will move to avoid the predator. In the presence of latency, this causes the predators to move to the *least likely prey location*. Moreover, if a prey were to know about network delays, it could use that information strategically to plot its own moves.

We propose a straightforward extension to the typical pursuit movement algorithm to account for latency. We add a *movement radius* to the algorithm; which is the region that must contain the prey. If we know the velocity of the prey—or at least its speed—and the network latency, we can compute a target area that must contain the prey.

The prey must be contained within this target region, as the maximum distance that it can move is given by the magnitude of its velocity times the latency. Thus, given a predator announcing a position, each predator will compute a target area and move toward that area. Further, we add a coordination strategy to the predators, in which they move to different positions within the target radius. This *dispersion strategy* is critical to ensuring that the predators do not move to the same position in the target area.

We implemented two dispersion strategies: the **Random Strategy--e**ach predator moves to a random radius (less than movement radius) and angle displacement within the region, thus ensuring that the predators randomly cover the target area. The **Doughnut Strategy--e**ach predator moves to a random location; however, we concentrate the predators in the outer region of the target area where the radius varies from the movRadius/2 to the movRadius. Since the prey will have likely moved away from the announced location by the time that the predators arrive.

Figure 2 shows the performance of our algorithm using the random strategy. The experiments were run under the same conditions as those in Figure 1, except with the random dispersion strategy. The x- axis is latency in ms and the y-axis is number of iterations necessary to capture the prey (again with f approximately equal to 16.7 ms). Comparing these data to those in Figure 1**,** this algorithm performs much better than the classic algorithm as latency increases: in general, the number of iterations necessary for capture decreased by about a factor of five.

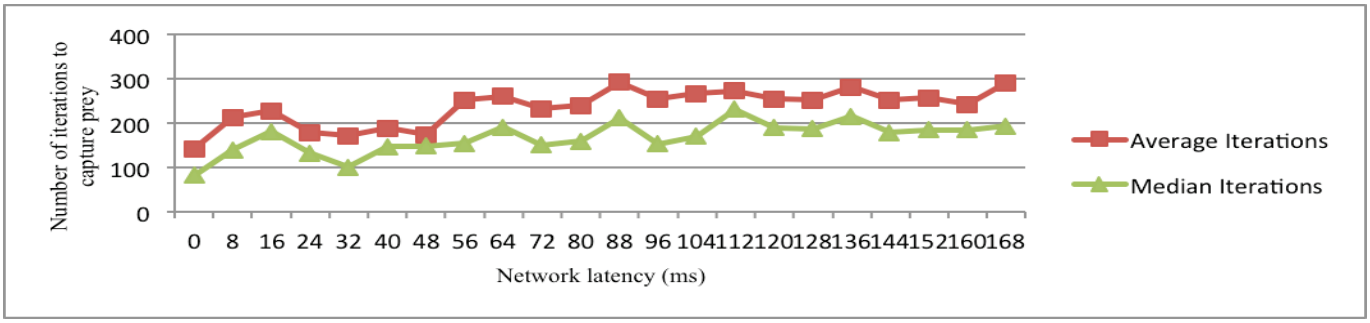

Figure 3: Effects of latency using target area "doughnut" strategy

The data for the doughnut strategy are show in Figure 3. Once again, the x-axis is latency in ms and the y-axis is the number of iterations necessary to capture the prey (*f* approximately equal to 16.7 ms). These experiments were run under the same conditions as those described in Figure 1, except that the doughnut strategy was used as the dispersion strategy. As observed with the random strategy, the doughnut strategy reduces the number of iterations necessary to capture the prey at a given network latency.

Conclusion

The coordination of predators in the classic pursuit problem is strongly dependent on communication among predators. Any latency in the messages among predators will cause degradation to coordination strategy performance, increasing the time it takes to capture a prey.

Rather than converge on the announced location of the prey, dispersing predators in a circular region around the prey gives superior performance under even large latency conditions. Two dispersion methods—random and doughnut—give roughly equivalent performance.

The strategies presented in this paper are just a start. We examined the effects of only two simple dispersion strategies. We have not explored packet loss, nor have we looked at latency that can vary widely over a short period. Perhaps most importantly, we need to investigate how obstacles affect the strategies.

Acknowledgments

We thank Stacy Birmingham for many useful comments on this paper. We acknowledge Grove City College's Swezey Fund for Scientific Research and Instrumentation for supporting this research.

References

Bererton, Curt. *State estimation for game AI using particle filters.* AAAI Workshop, 2004, 36 40.

Brown, Chris, George Ferguson, Peter Barnum, Bo Hu, and David Costello. "Quagents: A game platform for intelligent agents." *First Artificial Intelligence and Interactive Digital Entertainment Conference.* American Association for Artificial Intelligence, 2005. 9 14.

Claypool, Mark, and Kajal Claypool. "Latency and player actions in online games." *Communications of the ACM*, November 2006: 40 45.

Denzinger, Jorg, and Matthias Fuchs. "Experiments in learning prototypical situations for variants of the pursuit game." *PRoceedings of the Second Interantional Conference on Multiagent Systems.* AAAI, 1995. 48 55.

Heckel, Frederick W.P., and G. Michael Youngblood. "Multi agent coordination using dynamic behavior based subsumption." *Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.* American Association of Artificial Intelligence, 2010. 132 137.

Hladsky, Stephen, and Vadim Bulitko. "An evaluation of models for predicting opponent positions in first person shooter video game." *Proceedings of the IEEE Symposium on Computation Intelligence and Games.* Perth, Australia: IEEE, 2008. 39 46.

Isaza, Alejandro, Jieshan Lu, Vadim Bulitko, and Russel Greiner. "A cover based approach to multi agent moving target pursuit." *Fourth Artificial Intelligence and Interactive Digital Entertainment Conference.* American Association for Artificial Intelligence, 2008. 54 59.

Klaas, Mike, Tristram Southey, and Warren Cheung. "Particle based communication among game agents." *First Artificial Intelligence and Interactive Digital Entertainment Conference.* American Association for Artificial Intelligence, 2005. 75 80.

Korf, Richard E. "A simple solution to pursuit games." *Proceedings of the 11th International Workshop on Distributed Artificial Intelligence.* Ann Arbor, MI, 1992. 183 195.

Lamport, Leslie. "Time, clocks, and the ordering of events in distributed systems." *Communications of the ACM* 21, no. 7 (July 1978): 558 565.

Southey, Finnegan, Wesley Loh, and Dana Wilkinson. "Inferring complex agent motions from partiail trajectory observation." *IJCAI '07: Proceedings of the 20th Internation Joint Conference on Artificial Intelligence.* San Francisco, CA, USA: Morgan Kaufman, 2007. 2631 2637.

Weber, Ben G., Michael Mateas, and Arnav Jhala. "A particle model for state estimation in real time strategy games." *Proceedings of the Seven Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011).* Palo Alto, CA, USA: AAAI Press, 2011. 103 108.

Wittkamp, Mark, Luigi Barone, and Philip Hingston. "Using NEAT for continuous adaptation and teamwork formation in Pacman." *IEEE Symposium on Computational Intelligence and Games (CIG '08).* IEEE, 2008. 234 242.

Yannakakis, Georgios N., and John Hallam. "Evolving opponents for interesting interactive computer games." Edited by S. Schaal, A Ijspeert, A. Billard, S. Vijayakumar, J. Hallam and J.A. Meyer. *From animals to animates 8: Proceedings of the 8th International Conferen on Simulation of Adaptive Behavior (SAB 04).* Santa Monica,CA: MIT Press, 2004. 499 508.