BAYESIAN NETWORKS IN VIDEO GAMES

Jarett Cummings, Stephanie Elzer, and Gary Zoppetti
Millersville University
{jecummings,elzer,zoppetti}@cs.millersville.edu

ABSTRACT

This paper illustrates our attempts to utilize Bayesian networks as a method for implementing an artificial intelligence system for video games. Using these networks we attempt to control simple behaviors for game agents that can be applied to multiple game genres. We include the networks that were developed for certain behaviors and describe further projects that we intend to pursue. While not always completely successful, these behaviors provide some insight into the possibilities and problems of incorporating Bayesian networks into video games.

1. Introduction

The major focus of video game development has been on improved graphics, however, with advances in computing power and video cards that can handle these graphics the focus is shifting to AI. In most cases games feature non-playable characters (NPCs) that exhibit predictable and repetitive behaviors. Newer games require an improved intelligence system, which can keep up with the skills of the players. These games involve a large number of behaviors that are based upon a large variety of information about the world. It is not always possible, however, for the player or agent to know all of the information that may be required to perform certain behaviors and therefore it requires some sort of intelligence system for handling these uncertain situations. The use of Bayesian networks allows for rational decisions to be made without knowing all of the information and thus gives the illusion of an approximate human intelligence in the computer controlled characters.
Our primary, long-term goal on this project is to develop a general API which can be used by game developers to incorporate intelligent behaviors into games, based on Bayesian networks. We feel that it is possible to use Bayesian networks to develop an artificial intelligence system to handle certain situations that computer, and human, characters would face in a variety of game genres. This way the networks developed would not be unique for use with only one type of game. Thus we may be able to generate a generic infrastructure that can be used repeatedly. The other goal of this project is to develop networks and programs to handle multiple behaviors, such as path-finding. These networks would cause the appearance of not only human behavior, but also evolving behaviors in characters. These behaviors would not be game specific and therefore be able to be utilized in multiple domains. As a beginning step, we created networks able to handle domain specific behaviors, in order to observe generalities of the behavior in order to develop a domain independent API.

To help accomplish this goal we used a 3D graphics toolkit developed for Direct3D by research students under the direction of Drs. Zoppetti and Webster. The toolkit simplifies the development of games by providing a high-level object-oriented API for graphics, sound, and physics. The next step is to develop an API to ease the development of artificial intelligence for these games.

1.1 Goal

The primary goals of this project are to develop an artificial intelligence system that is able to adapt to a player's playing technique and be able to demonstrate learning to a certain degree. Ultimately we intend to develop a general API for use over a variety of game domains.

1.2 Background on Bayesian networks

Bayesian networks are based on a mathematical theory known as Bayes' Theorem, which is used to calculate the probability of an event occurring given a known related piece of information. Bayes' theorem states

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}.$$

Using this theorem we can calculate the statistical probability of events occurring even if we know very little about the world, which provides some information for reasoning under uncertainty [1]. This theorem allows us to determine a more realistic and probable assumption about the occurrence of A if we know that B has occurred. A medical example from Russell and Norvig's book, *Artificial Intelligence: A Modern Approach* illustrates this point.

Given a patient has meningitis; they suffer from a stiff neck 50% of the time. The probability of a person having meningitis is 1/50,000 and the probability of having a stiff neck is 1/20. Therefore, using Bayes' theorem we can calculate the probability of having meningitis given a stiff neck to be 1/5,000. This gives a much better assumption about the probability that a patient with a stiff neck has meningitis than just assuming it is 1/50,000 [2]. Therefore, the use of Bayes' theorem provides more reliable probabilities.

Bayesian networks are comprised of a set of variables and directed edges, with each variable having a

finite set of states. As Finn Jensen discusses, for each variable A with parents B1… Bn, there is a table of probabilities:

P(A| B1, …, Bn) [3].

Russell and Norvig discuss how Bayesian networks utilize the concept of conditional independence to scale up the power of the networks. Conditional independence allows the calculation of a variable even if knowledge of conditionally independent variables are unknown. For example, if variables X and Y are conditionally independent, and we wish to know **P**(X.Y.Z), then we can decompose this into **P**(X.Y|Z)**P**(Z) and using the idea of conditional independence we get **P**(X|Z)**P**(Y|Z)**P**(Z). By looking at this we are able to see that using these networks we can infer a probability by really only having knowledge of Z. This decreases the computational complexity of the network for assumptions like this because as the number of variables grows the representation would be O(n) as opposed to $O(2^n)$ [2].

Bayes' theorem and the Bayesian networks may not seem extremely powerful or useful at first but, when you consider the large number of situations where there is information that is missing or unknown, these networks demonstrate their power. These networks provide a reasonable probability that an event will occur, using uncertain data [2].

The use of these networks may provide an extremely beneficial aspect to video game artificial intelligence, which is the illusion of human intelligence in an NPC. As a human player plays a game they make mistakes but as the player learns more about the game and the world he adapts to make better decisions. This is the same with Bayesian networks. Under uncertain conditions the network will calculate the probability of a certain variable, which may be the wrong decision, but as more information is uncovered about the world the network is able to update the probabilities of other variables, causing the calculated variable to be updated, thus producing machine learning and an evolving artificial intelligence.

1.3 Related Work

There are games and ideas already on the market that utilize Bayesian networks as a way of intelligently making decisions and controlling and altering behaviors. The first example is Microsoft's Forza Motorsport, which utilizes what they call Drivatars. These Drivatars use Bayesian inference to allow the computer driven cars to monitor and mimic the driving style of the human players. This system is used as the underlying intelligence technique for controlling the computer drivers and allows for the computer to evolve and get better as the human player becomes better. This is done by using Bayesian inference to determine the line to drive that yields the highest probability of keeping the driver on course and finishing the race [4].

In an article titled "Teaching Bayesian behaviors to video game characters" Ronan Le Hy and his co-authors discuss the potential of using Bayesian techniques to allow computer characters to behave humanly and imitate player behaviors for Unreal Tournament. They discuss how they would use information about the character itself and the immediate surrounding environment to make decisions even in the presence of unknown variables. They are able to teach their bot new behaviors by analyzing human behaviors and allowing the bot to adapt its own behavior based on the human's state and chosen behavior [5]. Table 1 illustrates the results that they found when comparing bots that used learned behaviors, those that were hand specified, and the native bots to the game.

Table 1
Performance comparison on learned, hand-specified, and native Unreal bots (lower is better)

Recognition learned, aggressive	4.4
Recognition learned, cautious	13.9
Selection learned, aggressive	45.7
Manual specification, aggressive	8.0
Manual specification, cautious	12.2
Manual specification, uniform	43.2
Native (level 3/8) UT bot	11.0

It is clear that the learned behavior bots showed an advantage over the hand specified bots and even are comparable to the native bots of the game.

Both of these projects illustrate the feasibility and use that Bayesian networks can have as a form of artificial intelligence. The use of Bayesian nets for machine learning would allow more games to have an evolving and adaptive type of intelligence system providing better game play and more realistic decision making in video games.

2. Our Project

2.1 Previous Work

During the fall 2007 semester a partner and I developed Bayesian networks to handle two different problems that are applicable to video games. We attempted to use these networks to demonstrate learning using some simple video game behaviors. To solve these problems we used the Netica API [6] in order to implement, update, and perform the network calculations.

The first problem that we attempted using a Bayesian network was based on a problem presented in *AI for Game Developers* by David Bourg. In this problem we use a Bayesian network to determine the probability of whether an agent should open a treasure chest. In this problem the agent has experience opening a number of other treasure chests and has recorded information about the chests already opened, including how many were locked, had treasure, or were trapped [7]. This is a very simple network and we did not change it much for our uses. In our problem, the agent is provided with a

T

training set of chests of whatever size and using this training set gathers information about how many chests have treasure, are trapped, etc. The agent then proceeds to open a set of 50 other chests using the network to decide whether or not the agent should open the chest. The only information the agent knows about these 50 chests is whether or not the chest is locked or not. The network that we used for this problem is illustrated in Figure 3.

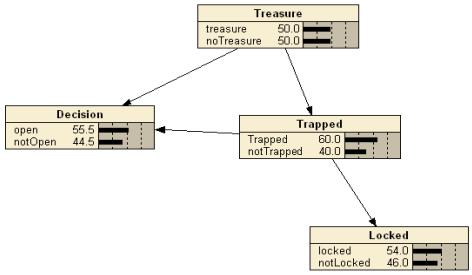
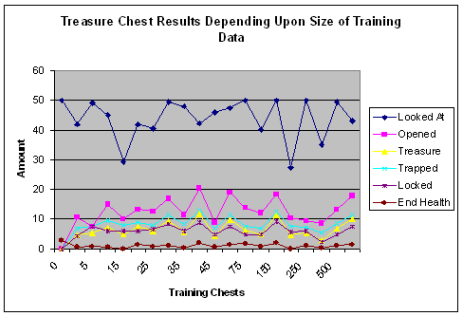


Figure 3 Example Bayesian network to determine whether or not to open a treasure chest.

When the agent starts to open the chest it had a health of 3. Each trapped chest it opened caused it to lose one health point, and for every chest that it opened that had treasure it gained one health point. The decision that we obtained from the network was the probability that the agent should open the chest. This was then considered along with the amount of health that the agent had. If it had high health then it was more likely to open the chest because it could afford to incur damage. Next we ran the simulation multiple times for each training data size and recorded the average number of chests considered, number opened, total treasure, trapped, and ending health. This data is illustrated in Table 2 and Graph 1.

Table 2
This table illustrates the average results ~~based on~~of 100 tests for each of the training set sizes ~~o~~pening 50 chests.

Training Chests	Looked At	Opened	Treasure	Trapped	Locked	End Health
0	50	0	0	0	0	3
1	41.9	10.6	4.5	6.9	4.4	0.6
5	49.1	7.6	5.5	7.6	7.6	1
10	45.2	15	7.1	9.5	6.1	0.7
15	29.3	10.1	4.9	7.9	6	0
20	41.8	13.1	7.4	8.9	6	1.5
25	40.6	12.7	6	8.1	6.6	0.9
30	49.4	17	9.6	11.4	8.4	1.2
35	48	11.5	5.8	8.4	6	0.4
40	42.2	20.3	11.7	12.8	8.8	1.9
45	45.8	8.8	4.4	6.9	4.9	0.5
50	47.3	18.9	9.7	11.4	7.6	1.3
75	50	13.9	6.2	7.5	4.8	1.7
100	40.1	12.2	4.9	7	4.9	0.9
150	50	18.3	11.4	12.4	9.3	2
200	27.4	10.3	4.6	7.6	5.6	0
250	50	9.4	5.4	7.3	5.9	1.2
300	35.1	8.6	2.9	5.5	2.4	0.4
500	49.5	13.2	6.8	8.5	4.8	1.2
1000	43.2	17.7	10	11.6	7.5	1.4



Graph 1 This graph illustrate the data from Table-2 ; Data should have gotten consistently better as training data size increased.

Unfortunately, as the data clearly illustrates this network did not work as well as we had hoped. As the training sets got larger, it would be logical to assume that the results would be consistently better than the tests performed with smaller training sets. The results, however, illustrate there was no consistent progressive pattern of learning; instead the results seem to be completely random and illogical. These results differ from the work results obtained by Bourg. Their problem, however, focused on a network used to make a single decision, while our network was developed to handle a series of decisions. By doing this we had hoped to observe learning as the amount of training data increased. We plan to continue work on this project to understand the results that were obtained and produce a more effective network or algorithm.

The second problem was a network to control the movement of an agent in order to provide the agent with the best path-finding ability to achieve its goal. The agent was present in a graphical world along with both a box that contained something beneficial to the agent (the "good box") and a box which contained something detrimental to the agent (the "bad box"). All three would be thrown randomly into the world and the agent was considered successful if it reached the good box and considered to fail if it touched the bad box. There was also a third alternative in which the agent simply took too long to touch either of the boxes. The goal was to help the agent efficiently reach the good box while avoiding the bad one. Figure 1 illustrates the network that we developed for this problem. We considered the positions of the agent and the boxes in only two dimensions, x and y. In our world the agent was privy to the x and y distances both between itself and the good box and also itself and the bad box. In the network we considered the range that the difference between the box and the agent fell into. If the distance was greater than the threshold, the effect of that box was less than if the distance was less than the threshold. Therefore, when the agent was within the good box's threshold the attraction was stronger and when the agent was inside the bad box's threshold the repulsion was stronger.

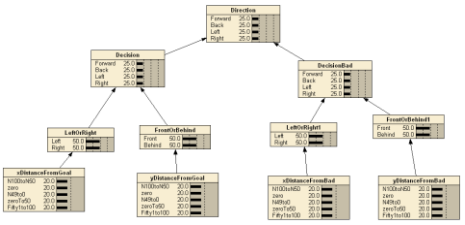


Figure 1 Example Bayesian network to determine the direction for an agent to travel based on location relative to a good and bad box.

Based on each of these measurements the network would first obtain probabilities for the best direction for the agent to travel for each box and then, based upon those probabilities, the network would calculate the probabilities that the agent's best move would be forward, backward, left, or right. For each probability that was ultimately calculated by the network, the agent would have that probability of traveling in that direction; therefore the agent did not always travel in the direction that had the highest probability returned by the network. While this seems illogical at first, this would sometimes allow the agent to still move even when the directions suggested by the two boxes were in opposite directions. This was an attempt to limit the amount of times the agent would become stalemated and not move. The agent would perform ideally when the two boxes were on opposite sides of it, because this was the easiest decision to calculate and the agent would head straight for the good box. The most common case in which the agent became stalemated was when the bad box was aligned directly between the agent and the good box. In most of the other cases, where the agent and the boxes were not aligned, the agent would usually reach the good box with it just taking a longer period of time than would have been desired. We tried a couple of different approaches in order to try to get the agent to reach the good box every time and in a timely fashion. The first idea was to adjust the thresholds at which the forces of the boxes got stronger. This way the agent would be more attracted to the good box at a greater distance and set the threshold for the bad box quite low so the repulsive force of the bad box was only strong when the agent got very close to the bad box. This way the agent could move more by being guided by the good box and really only be effected by the bad box when it got too close. This did work to an extent. The second change made to help the network better guide the agent was to change the network so that it altered the effect of the hazard. Instead of the hazard repulsing the agent back in the opposite direction it would shift the agent along side of it. This prevented the agent from getting stuck or being conflicted about which way to go. As a consequence the agent's time to the goal was greatly reduced and the agent's movements were less hesitant. By using this method we observed the desired behavior of having the agent take curved paths to the goal in order to avoid the hazard. Using this new network we observed the results displayed in Table 3. As is illustrated, the success rate of the newer network was approximately 93%. However, there were trials that succeeded almost every time. Considering the simplicity of the network the results achieved were very positive.

Con formato: Sangría: Primera línea: 0"

Con formato: Fuente: 8 pto

Table 3
This table illustrates the results for 10 tests letting the agent try to reach the goal 50 times per test.

Test	Successes	Failures
1	47	3
2	49	1
3	48	2
4	46	4
5	45	5
6	49	1
7	45	5
8	45	5
9	44	6
10	46	4
average	46.4	3.6

The standard algorithm used for path-finding in most games is A*. Bayesian networks, however, provide some benefits over A*. Unlike the A* algorithm, it is not necessary to maintain the previous states of the world using Bayesian networks. This cuts down on the memory required to use the algorithm, because we are just updating the network as the world or our knowledge changes. Also while a good heuristic can allow for quick computations, a mistake or a poor heuristic can lead to exponentially larger computation times with A* [2]. With a decent Bayesian network these mistakes can be easily rectified with little extra computation time. Because these networks are based on probabilities, they may not always make the same decision under the same circumstances which provides the added demonstration of intelligence. The networks are able to evolve and use the past as a reference in order to make better, more reasonable decisions.

2.2 Current and Future Work

During the spring 2008 semester we plan to continue using Bayesian networks to support an artificial intelligence system for implementing learning in video games. There are also additional behaviors that we are interested in studying and determining if any learning occurs. The first of these behaviors is one that would be especially useful for strategy war games. Based on known information about the world, including the player's army strength, opponent's strength (this may be known or estimated dependent upon the amount of exploration that has occurred), the strength of the player's defenses, the diversity of troops, the resources of the player, etc. These factors will be analyzed in a Bayesian network, which will ultimately return the probability of making a successful attack. The network could also be used to

determine the minimum number of troops that are needed in order for the attack to be considered a success. This way it will not always be necessary for the player to launch a full assault on the opponent and lose the whole army. This project could possibly yield an intelligent system for computer controlled players to utilize. The network would change as the information about the armies and the world changes. The network would create the illusion of an intelligent player that is capable of changing strategies in order to counter the techniques used by human players.

The second problem that we are interested in investigating would be especially applicable to RPGs to help characters make decisions with regard to ~~objects and~~fighting other characters in the environment. This idea is inspired by the example by John Laird and Michael van Lent in a presentation presented at a game developers conference in 2005. The network would use variables such as the characters current health, ~~and~~ strength, and defense and the health, strength, defense, and alignment (ally, enemy, or neutral) of the ~~object~~ character and determine the appropriate reaction that should be performed. These reactions would include attacking, defending, or fleeing [8]. This way the character is able to decide whether or not to fight and if it will fight, the best course of action against other characters in the environment. Now the network may not always determine the best reaction but as the character continues further and interacts more with the world it will learn better how to react to certain objects in the environment. This type of network could be used to control behaviors of the NPCs in RPGs and thus make them more robust characters. In this case the characters would not be so repetitive and monotonous and may be affected by more than just the player. This would provide better game play and much better game replay ability.

These projects provide insight into the power and abilities that Bayesian networks provide for use in artificial intelligence. They could at least theoretically be implemented in a variety of genres to handle a wide variety of behaviors. They not only allow for decisions to be made about these behaviors under conditions of uncertainty but as they are updated and the network is privy to more information about the world, they can demonstrate the characteristics of machine learning. These allow for the creation of video game characters that can evolve and match players' intelligence and also an intelligence system which is capable of being used across domains. These behavior networks provide examples for generalizing behaviors for use in our reusable API.

3. Conclusions

While the use of Bayesian networks for artificial intelligence systems is currently very limited, the benefits and advantages that they could provide seem enormous. The ability for these networks to provide inferences about certain behaviors and decisions under uncertain conditions allows for an all new type of computer intelligence. The ability for the machine to make beneficial decisions based on probabilities and update and evolve those decisions allows for players to face an intelligence system that can learn and mimic their playing style. This allows for these networks to be used for artificial intelligence over a wide variety of game genres. One of the limitations to using these networks is the necessity of having software for the networks themselves. While the networks may be fairly simple for simple behaviors as behaviors become more complex so do the networks. It will require the maintenance of more variables and could possibly use multiple networks. This complexity could be a downfall not only in terms of computational power but also in terms of programming complexity. However, with recent advances in multiple processors and the trend toward multi-core processors, the use of large networks becomes more feasible. Problems which were previously approximated using simple heuristics may become more manageable using Bayesian networks. These networks are ideal for game situations that rely greatly on statistics and are ideal for simple behaviors being performed under uncertain conditions.

4. Acknowledgements

Kevin Workman contributed to this work during the fall 2007 semester. The Direct3D toolkit was developed by the research students of Drs. Webster and Zoppetti.

References

[1] B. Coppin, *Artificial intelligence illuminated* (Boston, MA: Jones and Bartlett, 2004).

[2] S. Russel and P. Norvig, *Artificial intelligence: a modern approach.* 2nd ed. (Upper Saddle River, NJ: Pearson Education, 2003).

[3] F. Jensen, *Bayesian networks and decision graphs* (New York, NY: Springer-Verlag, 2001).

[4] Forza Motorsport 2: Artificially Intelligent, http://forzamotorsport.net/news/pitpassreports/pitpass36-2.htm (2007).

[5] R. Le Hy, A. Arrigoni, P. Bessière, & O. Lebeltel, Teaching Bayesian behaviors to video game characters, http://cogprints.org/3744/1/lehy04.pdf (2004).

[6]Netica APIs, Norsys, Website, http://www.norsys.com/netica_api.html

[7] D. Bourg & G. Seeman, *AI for game developers* (O'Reilly, 2004).

[8] J. Laird & M. van Lent, Machine learning for computer games, *Game Developers Conference*,

http://ai.eecs.umich.edu/soar/gdc2005 (2005).