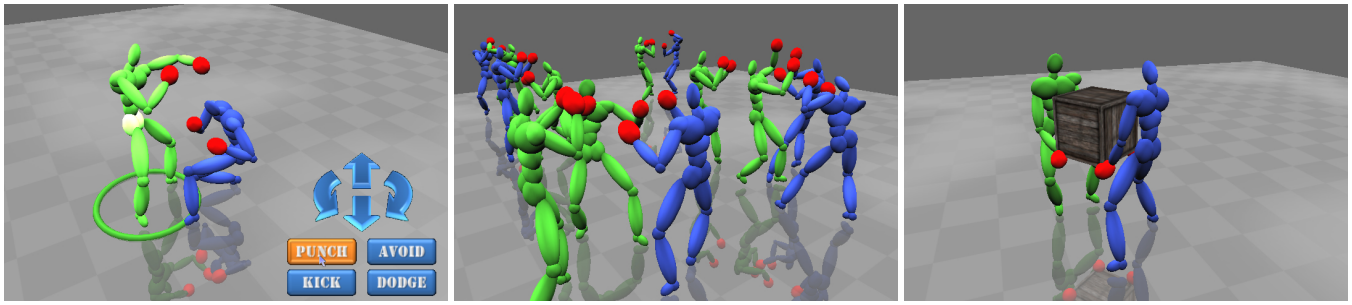
Simulating Interactions of Avatars in High Dimensional State Space

Hubert P.H. Shum* Taku Komura†
Institute of Perception, Action and Behavior
School of Informatics
University of Edinburgh

Shuntaro Yamazaki‡
Digital Human Research Center
National Institute of Advanced Industrial
Science and Technology, Japan



(a) (b) (c)

Figure 1: *The interactions of articulated avatars are generated by maximizing the reward function defined by the relative pose between avatars, the effectiveness of actions, and/or user-defined constraints. This framework of synthesizing character animation is efficient and flexible enough to make a variety of practical applications including (a) interactive character control using high-level motion descriptions such as punches, kicks, avoids and dodges, (b) real-time massive character interactions by a large number of automated avatars, (c) collaborative motion synthesis such as carrying large luggage by two persons.*

Abstract

Efficient computation of strategic movements is essential to control virtual avatars intelligently in computer games and 3D virtual environments. Such a module is needed to control non-player characters (NPCs) to fight, play team sports or move through a mass crowd. Reinforcement learning is an approach to achieve real-time optimal control. However, the huge state space of human interactions makes it difficult to apply existing learning methods to control avatars when they have dense interactions with other characters. In this research, we propose a new methodology to efficiently plan the movements of an avatar interacting with another. We make use of the fact that the subspace of meaningful interactions is much smaller than the whole state space of two avatars. We efficiently collect samples by exploring the subspace where dense interactions between the avatars occur and favor samples that have high connectivity with the other samples. Using the collected samples, a finite state machine (FSM) called Interaction Graph is composed. At run-time, we compute the optimal action of each avatar by min-max search or dynamic programming on the Interaction Graph. The methodology is applicable to control NPCs in fighting and ball-sports games.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Optimal Control, Human Animation

1 Introduction

Intelligent computer-controlled avatars are essential in computer games and animation. In many computer games, users can usually control an avatar to interact with other computer-controlled avatars. The intelligence of the computer-controlled avatar is important as it can affect the quality of the game. On the other hand, background characters in computer animation are also usually controlled by the computer. If their movements are unrealistic due to their poor intelligence, the animator needs to manually edit them, which will result in a huge amount of extra cost.

Traditional techniques such as decision trees and flocking have been used to control such avatars. However, those techniques can only generate reactive movements, and cannot realize strategic movements that benefit the avatars in the future.

Reinforcement learning enables real-time optimal control of avatars. It has been used to control pedestrians to avoid other obstacles/avatars walking in the streets [Ikemoto et al. 2005; Treuille et al. 2007], control a boxer to approach and hit the target [Lee and Lee 2004], make the transition of actions by the user-controlled avatar smooth [McCann and Pollard 2007] and train a computer-controlled fighter in computer games [Graepel et al. 2004]. However, there are two problems that we face when we try to use reinforcement learning to control human avatars intelligently when they are interacting with another avatar.

First of all, the state space is too large. The state space increases exponentially proportional to the number of parameters. Parameters

*e-mail: hubert.shum@ed.ac.uk
†e-mail: tkomura@inf.ed.ac.uk
‡e-mail: shun-yamazaki@aist.go.jp

such as the action the avatar is undertaking, its body position and orientation, and the timing to launch the action are going to form the state space. The number is going to double if there are two avatars. As a result, it is difficult for existing adaptive learning techniques such as Q-learning [Watkins 1989] to explore the whole state space to search for optimal policies.

Another problem is that the way the people behave change according to various factors such as their mood, habits, and preferences of actions; however, previous animation techniques used "on-policy" [Sutton and Barto 1998] reinforcement learning methods, which require the system to be retrained in case the reward function is changed. For example, in boxing, there are boxers called infighters who prefer to fight aggressively in short distance, and use punches such as upper cuts and hooks more. On the contrary, there are outboxers, who prefer to stay away from the opponent and as a result, prefer to use straight punches which are effective in long distance. If we train a virtual boxer by an on-policy reinforcement learning approach, it will not be able to compete well with other fighters who have different styles of fighting. The system needs to be pre-trained for various types of fighters, and the policy needs to be switched according to the type of the opponent, which will be very computationally costly.

In this research, we make use of the fact that the subspace of meaningful interactions is much smaller than the whole state space of two avatars. We efficiently collect samples by exploring the subspace where dense interactions of the avatars exist and favoring samples which have high connectivity with other samples. Using the samples collected, a finite state machine (FSM) called Interaction Graph is composed. The Interaction Graph is a Motion Graph of two avatars. In order to control the avatar in an optimal way, a min-max search / dynamic programming is conducted on the Interaction Graph.

We can compute motions similar to those of the optimal policies, although we plan motions on only a subset of the whole state space. As the way the space is explored is independent of the reward function for strategy making, we can also recompute the policy of the avatars in run-time based on the user's preference. We can simulate various activities by two avatars such as fighting, chasing, playing sports, or carrying luggage together. Our method can plan strategic movements for Non-Player Characters (NPCs) in 3D computer games. For example, we can control virtual fighters in boxing games (Figure 1(a)), or the background crowd moving or fighting with each other in computer animations (Figure 1(b)), or avatars collaboratively working, such as carrying a box (Figure 1(c)).

2 Related Work

Controlling human avatars intelligently in real-time applications such as 3D computer games has become important as the storage and computational power of computers has increased. Such avatars are usually controlled by Finite State Machines (FSM), in which each individual state represents the status of the avatars. The FSMs are usually manually designed, and the number of states and transitions are therefore limited.

A more complex version of such a FSM is the Motion Graph [Gleicher 1998; Lee and Shin 1999; Arikan and Forsyth 2002; Lee et al. 2002; Kovar et al. 2002], which is automatically generated from the captured motion data. The states and edges represent postures and short motion clips, respectively. Dynamic programming can be used to control the avatar intelligently in the Motion Graph.

When simulating the interactions of multiple avatars, the problem becomes far more difficult than controlling a single avatar. Park et al. [2004] creates a two-avatars version of the Motion Graph, and uses HMM to switch from one clip to another. This approach has a problem when collecting the data; as the motions of two persons need to be captured simultaneously and special arrangements are needed with the equipment. Capturing various combinations of motions is also very time consuming as it is much more difficult to direct two actors than one.

Recently, therefore, the problem of composing scenes of multiple avatars based on singly captured movements has attracted researchers. For example, Liu et al.[2006] creates such scenes by alternately computing the motions of individual character by using spacetime constraints. Movements which involve dynamics and physical constraints, such as a mother pulling the hand of a child can be created by this method. Shum et al.[2007] generates scenes of continuous interactions by defining an objective function which represents the benefit of each avatar, and selecting the optimal motion in the expanded game tree by min/max search. The problem with this method is that the computational cost is exponential, and therefore it cannot be applied to control avatars in real-time applications when the depth of the search is large.

If we want two avatars to intelligently interact with each other, reinforcement learning is a suitable approach, as it can find the optimal policy of two avatars while simulating their interactions. More specifically, at each time step i, suppose the avatar selects an action and gets a **reward** defined by r_i. The **optimal policy** π offers an action at every state that maximizes the following **return value**:

$$R = \sum_i \gamma^i r_i \tag{1}$$

where γ is called the **discount factor**, whose range is defined as $0 \leq \gamma \leq 1$.

The bottle-neck of using reinforcement learning is the large state space; the states of two avatars must be combined in order to express the condition of the avatars, which makes the problem difficult to be solved due to the so called curse of dimensionality. Researchers have proposed various methods to cope with the large dimensionality of the state space for using reinforcement learning in avatar control. Such researches include controlling NPCs of fighting games [Graepel et al. 2004], virtual boxers to fight with others [Lee and Lee 2004], and a walking avatar to avoid another avatar [Treuille et al. 2007].

Lee and Lee [2004] simulates a scene of two boxers fighting with each other with a precomputational approach. The boxers are trained by reinforcement learning so that they know the optimal way to approach and hit a target. The boxers are trained alone to find the optimal motions to approach and make a hit. There is, however, no concept of continuous interactions between two avatars during the training stage, although this is one of the most important issues for activities performed by multiple avatars. For example, there is no concept of defense, and therefore, the fighters will only try to approach and hit without taking into account the opponent's action.

Graepel et al. [2004] also uses reinforcement learning to train the computer-controlled player of fighting games. The system observes how the player fights with the computer-controlled player and learns the optimal policy to fight. In such fighting games, there are limited number of states such as "standing", "punching", "kicking", "defending" or "special attacks", and the effectiveness of attacks and defenses are judged based on the distance and its threshold, and the combinations of actions. As a result, the state space of the two avatars is actually not that large. When we handle continuous human interactions, we cannot radically quantise the state space because timing or the relative position / orientation greatly affects

the effect of each individual actions, and therefore, we cannot use their approach in our problem.

Treuille et al. [2007] simulates the interactions of two avatars approaching and avoiding each other. They used a near-optimal approach that represents the state of avatars by a weighted sum of basis functions. The parameters to control the avatars must be continuous as they are represented by the linear sum of the basis functions. As a result, the interactions are limited to avoiding moving obstacles or approaching avatars.

Up to now, to the best of our knowledge, no method based on reinforcement learning has been proposed to control avatars intelligently, when they are having dense interactions with other avatars.

3 Contribution

We propose a new off-policy learning approach that can intelligently control an avatar to collaborate / compete with another avatar in a huge state space. This is achieved by using criteria that favor states with good connectivity and more interactions. Our method is suitable for real-time interactive applications such as 3D computer games.

4 Outline of the Method

The procedure of our method can be divided into the preprocessing stage and run-time stage. The **preprocessing stage** proceeds as follows:

1. Capture the motions of a single person conducting the target motion and generate the action level motion graph structure out of the motion data
2. Explore the combined state space of two avatars by simulating the interactions of the two avatars and expanding the motion tree (Figure 2 left)
3. Generate the Interaction Graph of the two avatars and find the most appropriate movements of the avatars at each node by dynamic programming or min-max search. (Figure 2 right)

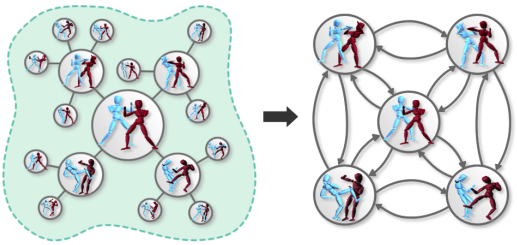


Figure 2: *The outline of the preprocessing stage: (left) exploring the state space by favoring states with more interactions and transitions to existing samples (right) an Interaction Graph created from the collected samples*

Then during **run-time**:

1. At each state, the corresponding avatar selects the precomputed optimal action
2. If the animator/user wants to change the policy/strategy of the control, the information in the lookup-table is recomputed by re-running dynamic programming or min-max search. This can be done in a few seconds, and can be run in background while simulating the interactions

5 Motion Capture and Simulation of Interactions

In this section, we explain how the motion data is captured and how we explore and collect sample data in the state space of two interacting avatars.

5.1 Data Acquisition and Preprocessing

We capture the motions of a subject singly acting, segment them into shorter semantic actions, classify them into different categories, and finally compose the Motion Graph [Arikan and Forsyth 2002; Lee et al. 2002; Kovar et al. 2002].

Capturing the motion of one person instead of those by two persons eases the collection of data. When capturing the motions of two avatars at the same time, there are many problems especially when using the optical motion capture system. Usually, such data requires a huge amount of post-processing, and the quality of the data is not as good as those when one subject is acting. Another advantage of capturing alone is that we can make various combinations of motions. If we limit the combinations to those of captured together, the type of samples will be limited, and as a result, the final animation will appear monotonic. We have captured motions of boxing and carrying luggage.

We next create the Motion Graph in the action level [Gleicher et al. 2003; Lau and Kuffner 2005; Kwon and Shin 2005; Shum et al. 2007], in which the minimum action unit is a semantic action such as a "step forward", "punch", "kick", or "fall down", instead of a posture or a short transitional motion clip. We do this by finding out the double supporting phases, segmenting the data there, and finally classifying the data automatically based on the trajectories of the arms and the feet. The details of this process can be found in Shum et al.[2007]. We also assume all the motion clips are correctly annotated.

5.2 State Representation

Here we explain how the status of two interacting avatars is represented; the approach is general enough so that any kind of interactions can be represented. The state space we consider here is composed of statuses when either avatar has finished an action and is about to start a new action. It can also be statuses when an action of the avatar is interrupted by the other avatar and is forced to start a new action.

Suppose we define the two avatars by A and B. We express the state when avatar A is about to select the next motion by the following vector: $I^A = (r, \theta_a, \theta_b, Next(M_a), M_b, F_b)$ where r is the distance between the two avatars, θ_a and θ_b are the angles made between the facing direction of avatar A and B, and the line connecting the two, respectively, M_a is the motion just finished by avatar A, M_b is the action avatar B is currently undertaking, $Next(M_a)$ is the set of actions that can be launched after M_a, and F_b is the frame number avatar B is at in motion M_b (Figure 3). The facing direction of each avatar is determined by its orientation of the head. We can define a state where avatar B is about to select an action in the same way: $I^B = (r, \theta_b, \theta_a, Next(M_b), M_a, F_a)$, where the variables are defined in the same way as those of avatar A.

5.3 Data Sampling

The data samples are collected by simulating the interactions of the two avatars and saving the state samples of I^A and I^B. During the simulation, when an avatar ends its action, it selects a motion

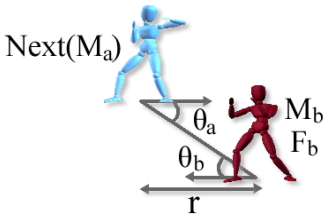


Figure 3: *Elements of the state that represent the relationship of the two avatars interacting*

among the next available motions in $Next(M_{\{a|b\}})$. Instead of continuously simulating the interactions along the time line as done in other adaptive reinforcement learning approaches, we collect the samples by selectively expanding the motion tree (Figure 4). The motion tree corresponds to the game tree in competitive environments. The nodes of the tree represent the states such as I^A and I^B, and the edges represent the actions that can be chosen by the avatar.

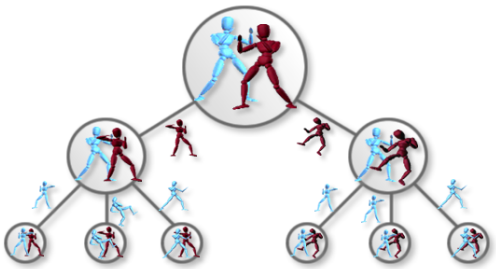


Figure 4: *An expanded motion tree. The red and blue nodes represent the state fighter A and B launch new actions, respectively. Each edge represents the action that has been selected by the fighter.*

The most important issue in this research is how to select a sample in the motion tree, and how to further expand the tree. We need to explore the space well enough to find the policy that can give the optimal solution to the avatar. On-policy methods such as SARSA [Rummery and Niranjan 1994] have been used to train the computer-controlled avatars [Graepel et al. 2004]. In on-policy methods, the samples collected are dependent on the reward function. Such methods will selectively explore the subspace which gives better rewards. For the simulation of two avatars interacting, the user might prefer to change the behavior of the avatars by adjusting the reward function. In such a case, if we use on-policy approaches, we need to retrain the system all over again. Instead, therefore, we use an off-policy approach here, in which the way we explore and collect samples is independent of the way the actions are rewarded. Using such an approach, the samples obtained can be used for different reward functions. As a result, we can change the behavior of the human avatar controlled by the system during run-time.

However, we cannot use well-known off-policy approaches such as Q-learning [Watkins 1989], in which the selection of the action is strongly affected by a random factor. This is because the state space is too large, and therefore, we need to efficiently explore the state space.

In this research, we explore the state space in a specialized way for the interactions of two avatars. We use an off-policy approach in which the criteria of selecting the space to explore are independent of the reward function. We need to explore the subspace that includes a lot of meaningful interactions between the avatars.

Starting from an appropriate state, we expand the motion tree. We favor states which (1) have high connectivity with other samples, and (2) results in dense interactions of the avatars.

Criterion (1) can be evaluated by counting the number of edges going out from the state that redirect the avatars to existing states in the state archive. Let us define this number by J_1. We adopt criterion (1) because we require a graph with high connectivity to increase the controllability of the avatar. A state of low connectivity is less useful as the chance to visit it is low and it is difficult to get back to other important states. If the number of states with low connectivity increases, the avatar needs to pass through a lot of extra states to finally start an effective action. This drops the performance of the avatar and eventually the animation will also appear unnatural. Through our experiments, we have found these criteria to be very important to collect good states.

Regarding criterion (2), the way to evaluate the amount of interactions between the avatars must be defined by the user: this can be simple if we are handling activities whose objective is clear, i.e., collaborative interactions such as carrying luggage together or competitive interactions such as fighting. The first way to evaluate whether there is an interaction or not is by setting constraints. For example, when carrying luggage together, the two avatars must not be too far away from each other as the luggage will fall down onto the ground. Motions leading them to get too close will also cause problems. Therefore, thresholds to keep the distance between the two avatars are set. Actions that violate such constraints can be considered as unsuitable and can be given low scores. For competitive actions such as boxing, the objective is to hit each other, or block / avoid those attacks which will otherwise cause the avatar to fall down. For boxing we evaluate the amount of dense interactions by the sum of attacks in the reaching distance and effective defense motions by both avatars. Let us define the amount of interaction by J_2.

At every state visited, we compute $J_1 + J_2$ of all the actions that can be launched. The children nodes are sorted based on the score, and the top 30% nodes will be further expanded. All the nodes explored in the motion tree are saved in the sample archive.

The exploration continues until either the number of samples in the archive reaches a maximum count, or until the number of newly created samples which do not duplicate with those already in the archive becomes smaller than a predefined threshold.

6 Interaction Graph

Based on the samples collected in the previous section, we compose a FSM whose states represent the interactions of two avatars. We call this FSM the Interaction Graph. Once the Interaction Graph is composed, by defining the objective function that evaluates the action chosen by each avatar, we can search for the optimal motion by using dynamic programming or min-max search. It is also possible to change the way each avatar behaves by editing the reward function and recomputing the policy in run-time.

In the following subsections, we first explain about composing the states of the Interaction Graph based on the collected samples (6.1), and then connecting the states by edges (6.2), and finally searching for the optimal motion at every state on the graph (6.3).

6.1 Creating States of Coupled Actions

After sufficient samples of interactions have been produced, the similar ones are grouped together using K-mean, and the states of the Interaction Graph are produced. A distance function D is defined as follows to calculate the difference be-

tween two samples $I_i = (r^i, \theta_a^i, \theta_b^i, Next(M_a^i), M_b^i, F_b^i)$ and $I_j = (r^j, \theta_a^j, \theta_b^j, Next(M_a^j), M_b^j, F_b^j)$:

$$D(I_i, I_j) = \begin{cases} \frac{|r^i - r^j|}{\sigma_r} + \frac{|\theta_a^i - \theta_a^j|}{\sigma_\theta} + \frac{|\theta_b^i - \theta_b^j|}{\sigma_\theta} \\ (\text{if } Next(M_a^i) = Next(M_a^j), M_b^i = M_b^j \text{ and} \\ |F_b^i - F_b^j| < F_e') \\ \infty \text{ (Otherwise)} \end{cases} \quad (2)$$

where σ_r and σ_θ are constants to normalize the effects of the distance and the rotation angles respectively, and F_e' is a threshold value.

Empirically, we found that $\sigma_r = 0.5$, $\sigma_\theta = \pi/6$ give good results. A state of the Interaction Graph is produced for every clustered group, which is represented by the average sample (Figure 5).

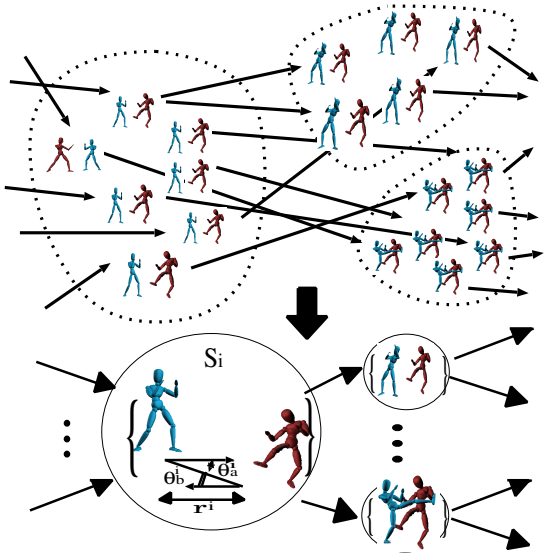


Figure 5: *The Interaction Graph: using the samples created by tree expansion, the state is composed by grouping similar samples.*

One important issue is how to determine the tolerance for grouping samples. If this value is too small, there are going to be too many states, and if this value is too large, there are going to be too many artifacts, such as foot sliding, sudden rotation of the body, and fast transition from one posture to another. Since foot sliding is the most noticeable artifact, we determine the threshold based on the amount of foot sliding that can occur.

6.2 Creating the Edges of the Interaction Graph

An edge in the Interaction Graph represents an action by the avatar to which the origin state belongs to. Recall one of the elements in the fight state, $Next()$ is the set of next available actions by the corresponding avatar. For every action in $Next()$, as a result of launching it, we might arrive to another state in the Interaction Graph. In that case, the two states are connected by an edge. However, since the graph is not covering the whole state space, there is a chance that there is no state in the graph after the action is launched. In that case, no edge is generated, and avatar A will not launch that action at this state. After scanning all the $Next()$ actions in all the states and the states are linked by the edges, the composition of the Interaction Graph is finished.

6.3 Search on the Interaction Graph

Once the Interaction Graph is created, we can do strategy-making by using dynamic programming or min-max algorithm. If the two avatars are collaboratively working, they will both select actions that maximize a common return function. Such a problem can be solved by dynamic programming. In case of competitive activities such as boxing, each avatar will try to maximize its own return value and minimize the opponent's return value. Such a problem can be solved by min-max search. In either case, we first need to define the reward function that evaluates the individual interaction. The rewards for each action at each state are precomputed, and therefore, its computational cost does not affect the run-time performance.

Let us define the reward function for carrying luggage as an example of collaborative activities. If we want to make the avatars proceed to a direction oriented ϕ to their average facing direction, we can compute the reward r_ϕ as follows:

$$r_\phi = w_\theta \theta^2 + w_d (d - d_p)^2 + w_v (v - v_\phi)^2 \quad (3)$$

where θ is the relative orientation of the carriers with respect to their partners, d is the distance with the opponent, d_p is the desired distance between the two, v is the average velocity of the two avatars, v_ϕ is the preferred average velocity of the two avatars, and w_θ, w_d, w_v are the weight constants for each term. We compute r_ϕ for eight different directions, as we would prefer to interactively control the two avatars to different directions during runtime. Using the reward function, we can compute the return value at each state by Equation 1 using dynamic programming.

For competitive interactions such as boxing and chasing, we use the reward in [Shum et al. 2007]. Let us briefly review the reward function that evaluates the action of fighters. It is composed of three criteria: (1) the relative orientation (θ) and distance (r) of the fighter from the opponent, (2) the damage the fighter has given to the opponent (D^+), and (3) the damage the fighter received from the opponent (D^-). The reward we use has the following form:

$$r_{box} = w_\theta \theta^2 + w_d (d - d_p)^2 + w_D^+ D^+ - w_D^- D^- \quad (4)$$

where d_p is the preferred distance by the fighter, D^+ is the damage that the fighter has given to the opponent, D^- is the damage received, and $w_\theta, w_d, w_D^+, w_D^-$ are the weight constants for each term.

In case of competitive activities, we do min-max search instead of dynamic programming, on the Interaction Graph. Let us explain how to do such a min-max search on the Interaction Graph. Assume $W_{i,j}$ is the reward of the j-th transition going out from state i. $W_{i,j}$ is computed by the reward function (Equation 4, in case of fighting), and it tells us how much the avatar earns/loses by launching the $j-$th action at state S_i. The values of $W_{i,j}$ are precomputed so that there is no need to evaluate the interactions of avatars during runtime. Suppose the best score the avatar can obtain from state S_i as a result of searching d levels ahead is $V_{i,d}$. Suppose we know $V_{i,d}$ for all the nodes already. In case we want to know $V_{i,d+1}$, we simply need to check all the edges going out from state S_i, sum the reward of the edge $(W_{i,j})$ with the return value of the state on the other end of this edge for depth d, and find out the edge that returns the largest (if it is a max node) or smallest (if it is a min node) value (Figure 6). The ID of the best next state is saved in $E_{i,d+1}$. The pseudo code of this procedure is shown in Algorithm 1.

The computational cost for finding the optimal path for min-max search is $O(N \times D)$, where N is the total number of states in the FSM and D is the maximum depth of the min-max search.

Algorithm 1 Min-max on the Interaction Graph

for $i = 1$ to N **do**
  $V_{i,0} = 0$ /* Initialize the score of 0-depth to 0 */
end for
for $d = 1$ to D **do** /* for each depth of search */
  for $i = 1$ to N **do** /* for each state S_i of the FSM */
    Scan the children of S_i
    s_{ij} is its index of the j-th child of S_i
    if S_i is a max node **then**
      $V_{i,d} \Leftarrow max_j\{V_{s_{ij},d-1} + W_{i,j}\}$
      $E_{i,d} \Leftarrow s_{ij}$ s.t. $max_j\{V_{s_{ij},d-1} + W_{i,j}\}$
    else /* S_i is a min node */
      $V_{i,d} \Leftarrow min_j\{V_{s_{ij},d-1} + W_{i,j}\}$
      $E_{i,d} \Leftarrow s_{ij}$ s.t. $min_j\{V_{s_{ij},d-1} + W_{i,j}\}$
    end if
  end for
end for

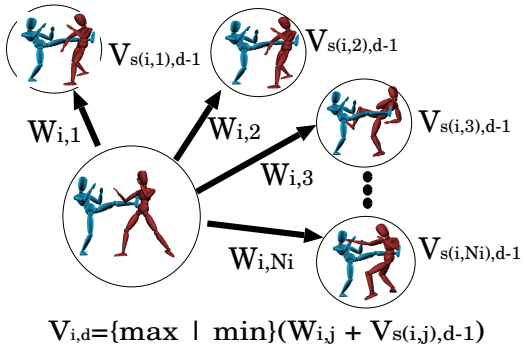


Figure 6: *Every node has a table which keeps $V_{i,d}$, which is the best score the avatar can obtain from state S_i as a result of searching d levels ahead. The number of out-going edges from S_i is N_i, the ID of the state on the other end of the j-th edge from state i is s_{ij}, the total number of states in the FSM is N, and the maximum depth of the min-max search is D.*

All $E_{i,d}$, which are the best states to transit to next at each state, are saved in a look-up table. This min-max computation has to be done only once for the whole graph unless the evaluation function is changed. This happens only when the user wants to change the parameters or sub-functions in Equation 4. Even if that is the case, it can be done in a very short time as all the parameters required to recompute them are embedded in the data structure of the transition.

7 Experimental Results

We have simulated scenes of fighting as examples of competitive interactions and scenes of carrying luggage together as examples of collaborative interactions. The readers are referred to the attachment video to see the animations. We have captured the shadow boxing of a boxer for 2.5 minutes, and a motion to carry objects for 2.5 minutes. Each motion was segmented into 137 and 167 actions, respectively, and was classified into different groups. A simulation based on the motion tree expansion was first done to compute and collect the samples. Using the obtained samples, Interaction Graphs of different categories were created.

7.1 Competitive Motion Synthesis: Kick Boxing

An Interaction Graph of 79,855 states and 3,392,297 edges was generated. The construction of the Interaction Graph took around 180 minutes using a Pentium IV 3.0GHz dual core PC.

In order to show the real-time performance of our system, we have implemented a game-style interface which the user can control an avatar to fight with the computer-controlled avatar (Figure 7). The user can give high level commands such as “move front / backward ”, “turn left / right ” “punch”, “kick”, “dodge”, and “avoid” to the avatar; the best action that belongs to such categories are selected in the Interaction Graph. The action of the opponent avatar is selected based on min-max search. All searches such as these can be undertaken just by looking on the table, and therefore, the computer-controlled avatar can react in real-time. A screen shot of such a scene is shown in Figure 1(a). The strength of the computer-controlled avatar can be adjusted by the depth of the search on the Interaction Graph; if we want to keep it weak, we can set it to 1, which means the computer-controlled avatar will only select the motion with the best immediate effect. It can be made stronger by increasing the depth of the search.

Figure 7: *Using the game style interface, the user can control an avatar to fight with the computer-controlled avatar by the Interaction Graph.*

We held matches between an avatar controlled by the Interaction Graph with three sorts of boxers; a human-controlled avatar, a computer-controlled avatar trained to hit a static object [Lee and Lee 2004], and a computer-controlled avatar by game tree expansion with the full character state space [Shum et al. 2007]. The avatar trained to hit the static object is considered to be the weakest, as it has no idea of defense. The avatar based on the game tree expansion is the strongest, as it evaluates all possible combinations by expanding the game tree; however, it cannot select an action in real-time when the depth level is ≥ 3. The results are shown in Table 1. The scores are calculated by subtracting the number of successful attacks by the Interaction Graph controlled avatar from the successful attacks by the opponent. It can be observed Interaction Graph controlled avatar is already too strong for the human controlled avatar to compete with when the depth level is over three. The avatar trained based on the static object also cannot perform well as it does not have any concept of defense. The Interaction Graph controlled avatar is weaker than that controlled by game tree expansion. The strength of the five level Interaction Graph avatar is about the same as the three level game tree boxer. This is because the game tree boxer's action is based on the precise simulation of the fight, as is done when forming the Interaction Graph, but before removing transitions or clustering. On the other hand, the choice of actions by the Interaction Graph includes quantization error. When

-	human	static	gametree 2	gametree 3
IG (1)	-15	34	-37	-53
IG (3)	23	54	14	-19
IG (5)	28	55	28	-4

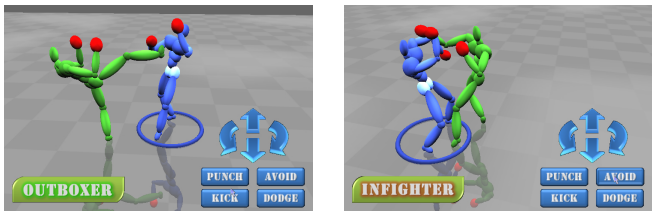
Table 1: *The score table of the matches between the Interaction Graph controlled avatar and the human player (human), an avatar trained based on static objects (static), and game tree search with expansion of two levels (gametree 2)and three levels (gametree 3). Each row shows the scores when the depth of the search on the Interaction Graph is 1, 3 and 5.*

the depth of the level increases, the quantization error starts to accumulate. Five levels of expansion is equivalent to a movement of 2.5 seconds in average. In activities such as fighting, each action is very quick and short; 2.5 seconds is already very long ahead in the future compared to the duration of each individual action. Therefore, we can say our results are satisfactory, taking into account that it can find out a near optimal solution by just looking it up on the table.

In order to check how many of the important states the Interaction Graph is covering, we have examined how many of the states visited by the game tree expansion approach are covered by the Interaction Graph. If this ratio is high, that means the Interaction Graph has a control policy comparable to that of game tree expansion. The result was 93%, which is very high, taking into account the size of the state space.

We can easily increase the number of boxers and create a scene where many avatars are fighting with their opponents (Figure 1(b)). The avatars are split into two teams, and each avatar fights with the closest avatar in the opposite team. During the simulation, sometimes other boxers get closer than the opponent the boxer is currently fighting with. In such case, the boxer switches the opponent. Therefore, in some cases there are scenes where one boxer is fighting with two or more fighters.

In order to show the effect of changing the reward function during run-time, we have enabled the computer-controlled avatar to change the style of fighting while fighting with the user-controlled avatar. When the style of fighting is switched to outboxing, if the user avatar approaches to the computer avatar, it will step backward or to the side to keep the distance between the two. On the other hand, when the computer avatar's style is switched to infight, it becomes more aggressive and always tries to hit the user fighter. Snapshots are shown in Figure 8. Regarding the parameters of the reward function, we used the same values as those in Shum et al.[2007].



(a) (b)

Figure 8: *Screen shots of the computer-controlled avatar switching its fighting style from (a) outboxing to (b) infighting in run-time.*

7.2 Collaborative Motion Synthesis: Carrying Luggage

An Interaction Graph of 128,804 states and 4,685,246 edges was generated. The construction of the Interaction Graph took around 200 minutes using a Pentium IV 3.0GHz dual core PC.

Rewards were set to move the avatars to eight locations around the avatars, and eight different policies were computed to move to each location. Once the policies are obtained, the user can interactively specify the direction the avatars should move, and the optimal motion is chosen from the corresponding policy. The user can control the avatars to move to arbitrary directions to avoid being hit by the ball. Screen shots of the avatars controlled to avoid the obstacles are shown in Figure 9.

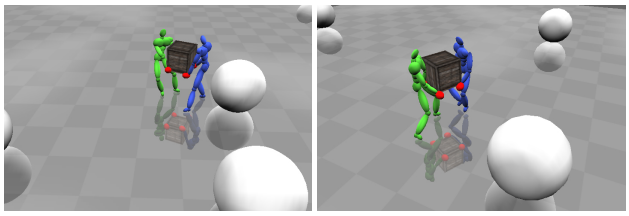


Figure 9: *Screen shots of the avatars controlled to avoid the ball while holding a box.*

8 Discussions

Due to the exponentially growing size of the state space with respect to the dimensionality, reinforcement learning could not be efficiently applied for dense interactions such as fighting, in which the avatars need to take into account the full status of the opponent. However, the subspace that meaningful samples exist is biased. By searching samples in the high density subspace, it is possible to compose a FSM called the Interaction Graph that enables intelligent control in real-time.

In our method, as the criteria to determine the subspace to explore is independent of the reward function that evaluates each action, the Interaction Graph can be used for different rewards, and it can be adjusted even during run-time. As a result, we can simulate the movements of different styles of interactions, such as infighters or outboxers who have different preferences for attacks and defenses.

Our method can be easily combined with existing FSM frameworks to control avatars in computer games such as wrestling. In such games, usually there is one FSM for each individual avatar. We can let the users freely control the avatars based on the individually prepared FSM when they are apart from each other, and let them go into the Interaction Graph only when the dense interactions start. Some game designers might prefer to manually design such motions so that they appear plausible. It is also possible to design a coupled state and insert it into an Interaction Graph.

The proposed method is deterministic; the action to be selected is determined based only on the min-max score computed over the interaction graph. The system can be easily switched to a probabilistic system, as done in Lee and Lee [2004]. We can set the probability that the avatar selects each action according to the min-max score, and use the Russian roulette approach to determine the action. We can also decrease such probability according to the number of times that that action has been selected at that state. For applications such as games, this approach can be a good method to randomly determine the actions of the computer-controlled avatar.

In our experiments, we do not include the discount factor when evaluating the actions at each state. This is because the depth of the search is only up to five levels. If a deeper search is required, we can use the discount factor when recursively computing $V_{i,d}$ in the Interaction Graph at Algorithm 1. In such a case, $V_{i,d}$ will be calculated by $V_{i,d} = \{max|min\}(W_{i,j} + \gamma V_{s(i,j),d-1})$ where γ is the discount factor.

There are some drawbacks in our system. Firstly, the criteria to determine the amount of interactions during the behaviors of the avatars must be specified by the user. As the user is only allowed to give an abstract idea of the interaction, this might not be a difficult task. Especially in case the interactions are competitive, an abstract idea of the way the two avatars compete can already become a good hint for the criteria.

Secondly, the user also needs to specify how to reward each action. This again depends on the nature of the interaction, and the user needs to adjust it to obtain a satisfactory scene. Since the topology of the graph is independent of the reward, the user can interactively edit the function and view the effect in the animation.

9 Conclusions

In this paper, we presented a real-time approach to simulate scenes in which multiple avatars are densely interacting with each other. We proposed a method to precompute the complex interactions of the avatars by favoring states that result in more interactions with the avatar and that have higher connectivity with other states.

Using our method, it is possible to cope with problems with high dimensionality, such as fighting. Our method can be used to control NPCs in 3D computer games as the optimal motion at every state is precomputed. We can even simulate various styles of interactions as the samples collected are independent of the cost function used to select the optimal action.

For future work, we are planning to further simulate the behavior of real humans when they are competing with each other. In scenes of competition, usually the person does not have full knowledge about the opponent, but gradually learns it through the interactions. The person also takes advantage of such a condition by launching fake actions to trick the opponent. By enabling the system to simulate such behaviors, it will be possible to create a virtual reality system that the athletes can use to train their skills and simulate matches with other athletes.

Acknowledgment

The authors would like to acknowledge the helpful comments of the reviewers. This project was partly supported by a CERG grant from the Research Grant Council of Hong Kong (RGC Reference No.: CityU1149/05).

References

ARIKAN, O., AND FORSYTH, D. 2002. Motion generation from examples. *ACM Transactions on Graphics 21*, 3, 483–490.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: assembling run-time animations. *ACM Transactions on Graphics 22*, 3, 702.

GLEICHER, M. 1998. Retargetting motion to new characters. *Computer Graphicsi Proceedings, Annual Conference Series (SIGGRAPH '98)*, 33–42.

GRAEPEL, T., HERBRICH, R., AND GOLD, J. 2004. Learning to fight. *Proceedings of Computer Games: Artificial Intelligence Design and Education (CGAIDE 2004)*, 193–200.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2005. Learning to move autonomously in a hostile world. *Technical Report No. UCB/CSD-5-1395, University of California, Berkeley*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3, 473–482.

KWON, T., AND SHIN, S. Y. 2005. Motion modeling for on-line locomotion synthesis. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 29–38.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 271–280.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. *Proceedings of 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 79–87.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. *Proceedings of SIGGRAPH'99*, 39–48.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3, 491–500.

LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2006. Composition of complex optimal multi-character motions. *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 215–222.

MCCANN, J., AND POLLARD, N. S. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics 26*, 3.

PARK, S. I., KWON, T., SHIN, H. J., AND SHIN, S. Y. 2004. Analysis and synthesis of interactive two-character motions. *Technical Note, KAIST, CS/TR-2004-194*.

RUMMERY, G. A., AND NIRANJAN, M. 1994. On-line q-learning using connectionist systems. *Technical Report CUED/F-1NFENG/TR 166, Cambridge University Engineering Department*.

SHUM, H. P., KOMURA, T., AND YAMAZAKI, S. 2007. Simulating competitive interactions using singly captured motions. *Proceeedings of ACM Virtual Reality Software Technology 2007*, 65–72.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

TREUILLE, A., LEE, Y., AND POPOVIC', Z. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics 26*, 3.

WATKINS, C. 1989. *Learning from Delayed Rewards*. PhD thesis, Cambridge University.