Evaluating and Tuning Motion Graphs for Character Animation

Paul S. A. Reitsma, Nancy S. Pollard

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-04-04
April 2004

Evaluating and Tuning Motion Graphs for Character Animation

Paul S. A. Reitsma*
Brown University

Nancy S. Pollard†
Carnegie Mellon University

Abstract

Realistic and directable humanlike characters are a continuing goal in animation. Motion graph data structures hold much promise for achieving this goal. However, the quality of the results obtained from a motion graph depends on properties of the graph that may be difficult for a user to control. This paper presents an approach to evaluating and tuning a motion graph data structure for a user's particular application. One key to evaluating a motion graph is to first embed it into the task domain. The goal of embedding the graph into the task domain is to capture all possible paths that might result from "playing back" the motion graph within the given environment. This paper describes an algorithm for accomplishing this embedding that is computationally efficient and also has the advantage of preserving the flexibility of the original motion graph. We then define an evaluation criterion on this embedding and describe a technique for tuning the motion graph based on this criterion. These techniques can assist the user in selecting an appropriate set of motion clips, adapting them for a given task, and answering the question of when sufficient motion is available for an application.

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

Keywords: animation, motion capture, motion graphs, evaluation

1 Introduction

One of our goals is to make realistic animation of human motion accessible to a broad range of users. Motion graphs have a particular appeal—the ideal vision is that a user can load a set of motion clips, automatically generate a graph structure, and instantly begin to direct the character using intuitive interfaces such as mouse gestures, full body demonstration, or specification of a set of constraints that should be satisfied.

However, the reality is often very different, and results obtained using such a system depend on many parameters. For example, consider the question of how much motion data to include in a motion graph. Including an excessively large amount of data can result in long computation times to construct the graph and to search for paths that meet user objectives. Insufficient data results in a character with limited capabilities; if the motion is inadequate, there may be important parts of the task domain that the character cannot reach. Another potential flaw is lack of variability in the character's behavior. For example, there may be only a single clip that allows the character to jump a large stream; if the character performs this same motion many times, its behavior will begin to seem overly scripted or robotic. A variety of tools have been developed to help the user annotate and edit a motion database to improve results (e.g., [Arikan et al. 2003] [Gleicher et al. 2003]). However, evaluating and tuning character capabilities is still substantially a trial and error procedure, especially for users who are less experienced with motion graph data structures.

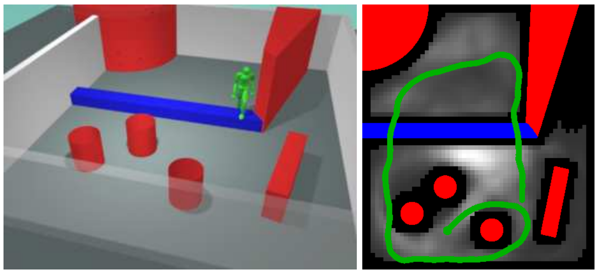

Figure 1: (Left) One of our test environments. (Right) Embedding our motion graph into this environment tells us what areas the character can and cannot reach. Here, areas that the character can reach are marked in various shades of grey. Black areas cannot be reached by the character given this motion graph. The green line traces a random walk through the embedded graph.

This paper describes a set of techniques that complement existing editing tools by providing a means to evaluate and tune character performance within a specific task domain. Evaluation and tuning are greatly simplified by embedding the motion graph into the task domain, because this embedding places the motion graph in a global reference frame and can reveal the extent to which the character will be capable of interacting with objects, avoiding obstacles, and achieving task goals within its environment. Section 3 describes a *resolution complete* technique for accomplishing this embedding. In other words, given infinitely fine resolution, all possible paths would be represented. As a result, our algorithm has the advantage that it preserves the flexibility inherent in the original motion graph. Section 4 then describes how weak points of the graph can be identified based on a local measurement of motion variability. It also describes how the embedded graph can be pruned in a way that maintains variability in the motion.

For example, Figure 1 shows some results for a motion graph constructed from motion clips that include walking and stepping over an obstacle. For our motion graph and environment, the figure on the right displays all of the regions that can be reached by the character. The lighter regions contain greater variability in the motion, where variability captures the number of ways in which the character can enter and depart from a given region.

2 Background

This work draws on research combining the use of human motion capture data with path planning. One approach that has been used to combine motion data with planning is to plan a trajectory in the low dimensional task space and then add motion detail to that tra-

*email: psar@cs.brown.edu
†email: nsp@cs.cmu.edu

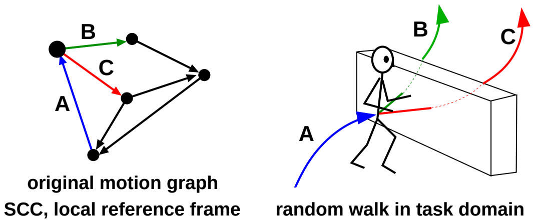

Figure 2: A motion graph may be constructed using a reference frame local to the character to preserve flexibility in the character's motion. However, driving the character through this motion graph in an environment with obstacles can result in dead ends that might be encountered by a local planner.

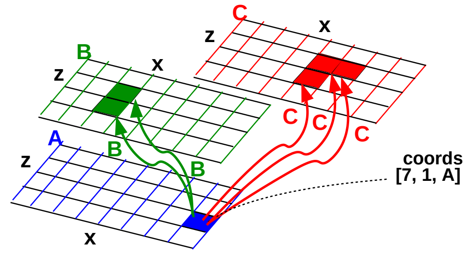

Figure 3: Example of forming links for an embedding. The task domain is horizontal position (x,z), and three motion segments are shown (A, B, and C). Links from element $[7,1,A]$ are formed by looping over all segments that follow from A and finding the footprint of destinations that can result from small modifications of the original motion.

jectory (e.g., [Shiller et al. 2001], [Pettré et al. 2003]). Kuffner and his colleagues [Koga et al. 1994], [Kuffner et al. 2002] and Kalisiak and van de Panne [Kalisiak and van de Panne 2001] have also applied probabilistic planning techniques directly in the high degree of freedom configuration space of humanlike characters and humanoid robots by identifying a set of contact conditions or poses that help to constrain the search space. These algorithms are most applicable when the character or robot is free to move within its workspace. We assume that our characters are not completely free to move within their workspace, but are instead constrained to follow prerecorded motion segments that are connected to form a motion graph.

Motion graphs have been a topic of intense investigation in the past few years (e.g., [Molina-Tanco and Hilton 2000][Lee et al. 2002][Kovar et al. 2002a][Arikan and Forsyth 2002][Li et al. 2002][Kim et al. 2003][Arikan et al. 2003]). Motion graphs are often used in situations where the user has high-level control over the character's motion, and it can be convenient to construct them without reference to a particular environment. Expressing position and orientation of the character in a local reference frame that will be later mapped into the environment provides the flexibility needed to use a small collection of motion drawn from a confined capture area to direct the character through a large task space.

If character motion is driven by a local planner, however, the position of the character with respect to any obstacles in the environment becomes important. In particular, obstacles can cause local planning techniques to encounter dead ends when the motion graph is constructed in a local reference frame, even when the motion graph forms a single strongly connected component (SCC) (Figure 2).

An alternative approach to using a local reference frame for the character root is to embed the motion graph into the task domain. Finding an SCC within this embedding completely eliminates dead ends due to static obstacles. Work in this area has been pioneered by Lee and his colleagues in two different approaches [Lee et al. 2002][Choi et al. 2003]. In the first approach, the authors build the original motion graph using a coordinate frame fixed to the environment, which allows them to capture interaction with a stepstool and playground within a constrained area of motion [Lee et al. 2002]. Using a fixed coordinate frame has the advantage that motions will always be correctly oriented toward the objects in the environment and physical interactions with those objects will appear plausible. However, this approach greatly reduces flexibility in reusing the motion capture data, because even a motion that does not involve interaction with the environment is tied to the location at which it was originally captured.

In the second approach, Choi, Lee, and Shin [Choi et al. 2003] show how a motion graph can be embedded into the task domain by essentially "unrolling" the motion graph onto a "roadmap" constructed in the task domain. (See Kavraki and Latombe [1998] for an excellent description of path planning using roadmaps.) This approach has the advantage of producing even coverage of the task domain and can lead to very fast planning at runtime. However, it again reduces flexibility in how the original captured motion can be used, because the roadmap contains an embedding of only a small subset of the original motion graph; a single motion clip is associated with each edge of the roadmap.

Our first contribution over this previous work is to show how an *entire* motion graph can be embedded into a task domain. Embedding the entire graph into the task domain allows us to preserve both types of flexibility mentioned above: (1) the ability to reuse the same motion clip in different parts of the task space and (2) the ability to choose among different sequences of motion clips to traverse the same part of the task space. To make this process tractable, we use a grid-based approach (Section 3). Grid-based approaches have been employed in robot path planning (e.g., [Lozano-Pérez and O'Donnell 1991] [Latombe 1991] [Donald et al. 1993]), but these algorithms are designed to achieve a different objective: find a single good path for the robot to move from start to goal.

The second contribution of this paper is to describe some ways in which the embedded graph can be used to evaluate a character's capabilities and to tune the graph for improved runtime performance (Section 4). The primary evaluation criterion used in this paper is variability, a local measure of the number of alternative paths that can be taken through a region of the task domain. A number of researchers have developed models of variability in motion (e.g., [Pullen and Bregler 2002][Rose et al. 1998][Brand and Hertzmann 2000]). These models either separate style from motion content ([Rose et al. 1998][Brand and Hertzmann 2000]) or provide a statistical model of motion detail that can be applied to an initial sketch of the motion ([Pullen and Bregler 2002]). However, variability models in these papers are developed with the goal of creating appealing motion, not to evaluate an existing motion set as in the current paper.

Finally, we mention the work of Gleicher and his colleagues to develop tools for editing motion graphs [Gleicher et al. 2003]. Our work shares their goal of improving graph utility, for example to create poses that are "hubs," having a large branching factor. Our algorithms could assist the user in evaluating the utility of adding new motions to increase character capabilities or the utility of rearranging the graph by making different hub choices, for example.

3 Embedding into the Task Domain

One insight of this paper is that a motion graph data structure can be more easily evaluated once the entire motion graph has been embedded into a task domain. We assume the task domain to be the static portion of the character's environment. For a simple navigation task, for example, the task domain is three dimensional. The character can move about on the ground plane and can be oriented in different directions about the vertical axis.

We also assume that the motion graph has been captured without reference to a particular environment, although it may contain motions, such as jumping, stepping, or climbing over an obstacle, that can be transferred to different situations. Embedding, then, captures the interaction of the character with a particular environment. Embedding the motion graph into the task domain for a navigation task, for example, captures the character's location relative to obstacles in the environment and maps motions into specific task situations. For example, stepping motions in the motion graph used in Figure 1 become associated with crossing the blue obstacle in Figure 1.

The high level idea is to capture all of the ways in which the motion graph could be "unrolled" into the environment. In other words, starting from anywhere in the task domain and following paths in the motion graph, where could the character travel, and what is the character be capable of doing? The task domain is a continuous space, and so expressing all paths is not in general possible. However, this effect can be approximated using a grid based approach. Section 3.1 describes an efficient way to unroll a motion graph into a grid that has been placed over the task domain.

Simply unrolling the motion graph into the task domain is not sufficient, however. Our goal is to obtain a connected graph of finite size within the task domain through which the character could wander indefinitely. To achieve this goal, we need to identify situations where the unrolled graph connects back onto itself—in other words, situations where the character returns to exactly the same state. For this to happen, the character must return to the same node in the motion graph at the same location in the environment. In a continuous task domain, this situation would generally never arise. To force this situation to arise, we force the motion segment endpoints onto grid centers; each motion segment must originate and terminate at the center of a grid point.

Forcing motions to originate or terminate at grid centers requires some editing of the motion capture data. If sufficient editing is allowed, then a single segment may reach more than one grid center. In other words, each motion segment would connect to a "footprint" of destinations. To compute this footprint, we must model the amount that a given motion segment can be edited. How to model this allowable error is not yet well understood. Ideally, this model would capture editing operations that would have little effect on the user's perception of the motion (e.g., see [Reitsma and Pollard 2003]). In practice, we use a very simple model, where this footprint size is a linear function of the distance that the character travels.

3.1 One-step Unrolling into a Finite Space

Given the assumptions described above, embedding an entire motion graph into a task domain can be accomplished by (1) performing a one-step unrolling process at each grid element, (2) locally forming links between incoming and outgoing nodes, and (3) identifying the largest strongly connected component (SCC) within the resulting collection of links.

The one-step unrolling process forms all possible links out of each grid element. Figure 3 illustrates this process. The figure shows a simple task domain that is represented by two position coordinates, x and z. Grid elements are indexed by their (x,z) position and by incoming motion clip. The figure shows portions of the task domain for three incoming clips: A, B, and C. The highlighted element, for example, is located at an (x,z) position of $(7,1)$, and only motion clip A can enter this element.

To form all possible links out of a grid element, we first identify outgoing motion segments by examining the original motion graph. If we assume the motion graph in Figure 2, then incoming motion segment A can link to outgoing motion segments B and C. One-step unrolling from grid element $[7,1,A]$, then, will involve forming links from that grid element to grid elements within the footprints formed by playing motion clips B and C from the starting point $(7,1)$. The resulting links are shown in Figure 3.

Note that repeating this process just once for each grid element results in forming all possible links in the graph. All starting coordinates, all motion clips, and all of the editing variations that we model are considered. We also mention that although the character is only able to enter the highlighted grid element in Figure 3 along motion segment A, this grid element may have many incoming links at the end of the one-step unrolling process. The character may be able to enter grid element $[7,1,A]$ by playing motion segment A from several different locations, especially if the footprint of A is large.

After the one-step unrolling process has been performed for all grid elements, we find the largest SCC in the directed graph [Cormen et al. 2001]. This SCC is the desired embedded graph.

3.2 Constraint Checking

Many links will not be possible due to collisions with obstacles in the environment or other constraint violations (e.g., attempting to walk across a chasm). Constraint checking is done at the time the links are formed. We do not process grid elements where the character would collide with obstacles, because these locations cannot be part of the embedded graph. Similarly, links that terminate at grid elements that result in collisions are culled immediately. For links that pass this initial test, the entire path represented by each link is checked for collsions and other constraint violations. In Figure 3, if there is a wall blocking all of the outgoing links (as shown in Figure 2), these links will be removed from the graph at this time. Finally, an SCC is computed from links that remain.

We have implemented just two types of constraints in our examples. The first is to force the character to avoid obstacles. The second is an annotation constraint, where the environment is annotated to indicate regions where a specific type of motion should or should not be performed. This is the constraint type that was used to map stepping motion to the blue obstacle. This same procedure could also be used for more freeform annotations, such as placing a particular dance move only in front of the judges.

3.3 One-step Unrolling into Tiled Spaces

Figure 1 shows a confined space. However, the algorithm is also applicable in open spaces, or any other space that can be regularly tiled. Whenever a space can be tiled, the embedding algorithm is run on a single tile, and edges are allowed to wrap around and back onto that tile. For example, open space can be treated by selecting a reasonably sized rectangle and treating this rectangle as topologically equivalent to a torus.

The implementation for tiled spaces is straightforward. When the one-step unrolling process shown in Figure 3 is run, each link is checked to see whether it crosses a tile boundary. If so, destination grid indices are recomputed modulo the tile size. We store both the original destination and the "wrapped" destination to make it easy to properly align the motion clips as the character travels through the tiled environment.

3.4 Correctness of the Embedding Algorithm

Our embedding algorithm is resolution complete in the sense that a sufficiently fine grid will capture all significant variations in character paths that are possible given our motion graph, task domain, and footprint model. However, at practical grid resolution, some possible paths will be lost. Figure 4 shows an example. Consider

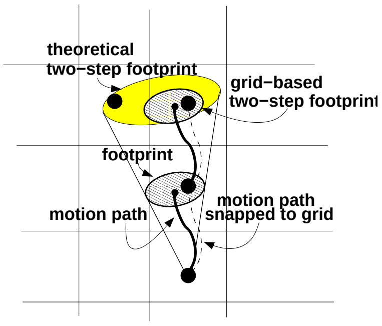

Figure 4: In our grid-based algorithm, the origin and destination of each motion segment is snapped to the centers of grid elements based on the footprint of acceptable edits to the motion segment. This figure compares how growth in the theoretical footprint of edits compares to the footprints actually used in the grid-based algorithm.

a straight walking motion that can repeat. Figure 4 compares two steps of computing a grid based footprint to the theoretical footprint after two steps. Some of the acceptable paths within this area are dropped when forcing the one-step path to the grid center. As the grid becomes finer, the amount of animation time that must elapse before the algorithm eliminates theoretically possible paths through the gridded task domain will increase.

The embedding algorithm can be made conservative by making connections only to grid centers within the footprint. If the algorithm is implemented in this way, no paths can be generated that are not possible given the motion graph, task domain, and footprint model.

We chose to use a slightly different approach, however, and always snap the end of a motion segment to the nearest grid center, regardless of whether that center is in the interior of the footprint. Our approach preserves the connectivity of the original motion graph. However, if footprint size is small compared to grid size, it can introduce paths that are not possible given our model. To avoid introducing such paths, grid resolution should be kept sufficiently small that all footprints will contain at least one grid point.

4 Evaluating/Tuning the Embedded Graph

We would like to use the embedded graph to identify weak points in the set of character capabilities. If weak points are found, additional motion or more forgiving criteria for selecting allowable transitions are needed.

One way in which an embedded graph can show weak points is that it may contain areas that the character cannot reach at all or can reach in very few ways. To capture this notion, we propose a local measure of variability that expresses the number of paths through a grid element as follows.

For task coordinates p and motion segment c, let $incoming(p,c)$ count the number of links that play back motion segment c and terminate at task coordinates p. Similarly, let $outgoing(p,c_j)$ count the number of links that play back motion segment c_j and leave from task coordinates p. Then variability can be expressed as

$$clipVar(p,c) = incoming(p,c) * \left(\sum_{c_j \in c.children} outgoing(p,c_j) \right) \tag{1}$$

$$variability(p) = \sum_{c \in C} clipVar(p,c) \tag{2}$$

where C is the complete set of motion segments and $c.children$ contains clips that follow from c in the original motion graph. The expression for clip variability is the product of the number of links entering the grid element indexed by p and c with the number of links leaving that same grid element. The expression for variability is the sum of clip variability measures over all motion segments and captures the number of different ways in which the character can pass through task coordinates p.

We use the term $coverage(p)$ to indicate whether the character can reach a location p in the task domain at all. In other words,

$$coverage(p) = variability(p) > 0 \tag{3}$$

A motion graph will have good coverage if $coverage(p)$ is true for a large percentage of points p within the task domain.

Coverage and variability can easily be visualized for low dimensional spaces. In higher dimensional spaces, a brushfire algorithm such as that used by Lengyel and his colleagues [1990] for fast path planning can be run on the grid that has been placed over the task domain in order to identify large areas that the character cannot reach at all or that have unacceptably low variability.

In many situations, sufficient motion is available, and a more pressing problem is to prune the embedded graph to a managable size for efficient planning. We propose a greedy local technique for pruning the graph that is simple, fast, and effective in practice. This local approach considers each set of task coordinates p independently and selects only the k best grid elements to preserve at p. The goal is to prune nodes that contribute least to the variability at p. By pruning nodes with the smallest number of connections, we hope to create a graph where all of the nodes can be reached in many ways.

For example, if we wanted to keep only the single best node at coordinates p, we would keep the grid element indexed by p and motion clip $c_m(p)$, with

$$c_m(p) = argmax\{clipVar(p,c) : c \in C\} \tag{4}$$

All other nodes associated with point p, along with their corresponding incoming and outgoing links, would be pruned from the graph. The strongly connected component algorithm would then be rerun to extract a new SCC from the pruned graph.

5 Results

Our primary test scene is a cluttered environment through which the character must navigate. Figure 1 shows a three-dimensional view of the environment and an overhead view of a random walk through the embedded graph. In this example, we consider the task domain to be three-dimensional, including the position of the character on the ground plane and its orientation about the vertical axis. The size of the room was 7m by 8m. The room was divided into grid elements with a resolution of 12.5cm, and a grid spacing of 15 degrees was used for the character's orientation about the vertical axis.

We began by forming a motion graph in the character's local reference frame. The motion set for this graph was 23 motions downloaded from the CMU motion capture database (mocap.cs.cmu.edu). These motions included walking, walking while turning, and stepping over an obstacle. These motions were relatively short clips and the total amount of motion was approximately 77 seconds. Costs for transitions between motion segments were computed using the technique of Lee et al. [2002]. The map of transition costs was then processed to identify local maxima and to enforce a user-specified minimum clip length. When identifying transition locations that were local maxima, we used an objective function that attempted to preseve the largest possible number of connections to other motion segments, with each connection weighted by goodness of the transition; our goal was to preserve as much flexibility in the character's motion as possible. Enforcing a minimum

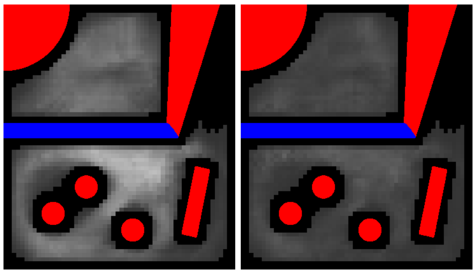

Figure 5: (Left) Variability of the graph after local pruning from 55 clips to 16 clips. (Right) Local pruning to 4 clips.

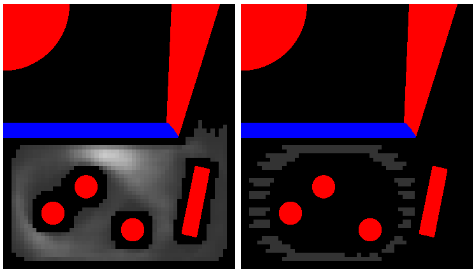

Figure 6: Global pruning by keeping motion segments having maximum variability when integrated over the entire task space is much less effective than local pruning. (Left) Variability when 36 clips remain. (Right) Variability when 8 clips remain.

clip length reduced some flexibility, but was important for keeping the number of links in the resulting motion graph small. Keeping the motion graph small reduced both memory use and computation time for the subsequent embedding process. Forming a motion graph in the character's local reference frame on our motion set resulted in an SCC with 55 nodes and 147 links. When playing back the motion, transitions between motion clips were smoothed by using quaternion splines to gradually close the gap formed at the transition point. No other cleanup was done in our examples. Most notably, we did not fix footskate artifacts, although that could easily be done as a postprocessing step (e.g., [Kovar et al. 2002b]).

The motion graph in the character's local reference frame was then embedded into the task domain as described in Section 3. The code was written in Java, and run on a 2.2GHz Xeon computer with 1.8GB of memory allocated to the Java executable at runtime. Simple collision checking was done by giving the character a uniform radius in the horizontal directions. The embedded graph had 198K nodes and 1.47M links, and required approximately 30 minutes to compute.

Local pruning as described in Section 4 can be used to reduce the size of the embedded graph to reduce storage or to speed planning. Figure 5 shows results after links with lowest variability were removed. The left figure shows variability for the embedded graph after each task coordinate p has been pruned from 55 incoming motion segments to 16 motion segments. The right figure shows results after pruning to 4 incoming motion segments. Nodes were pruned from 198K to 150K and 53K respectively, and links were pruned from 1.47M to 1.15M and 404K respectively. Compare these figures to that on the right of Figure 1, where all 55 incoming motion segments were retained.

We note that variability could also be used to do global pruning. In other words, we can identify the motion segment that has the lowest variability when integrated over the entire task domain and remove that motion segment completely from the database. Doing global pruning in this manner, however, does not produce good results, because motion segments that are important for connecting

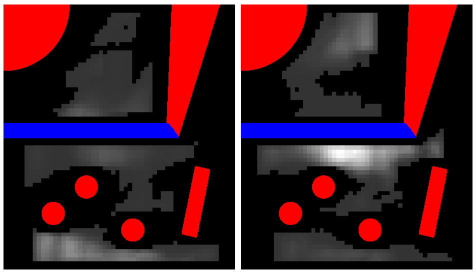

Figure 7: Variability for two separate character orientations. (Left) The character is facing the left side of the figure. (Right) The character is facing the right side of the figure. Areas where the shading is brighter in the left figure than the right figure, for example, are areas where the character has more motion choices when facing left than when facing right.

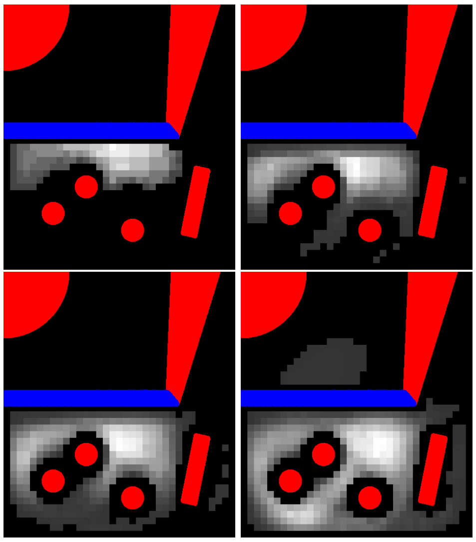

Figure 8: Areas from which the character can reach the area beyond the step in a given amount of time. (Top, left) 2 seconds. (Top, right) 3 seconds. (Bottom, left) 5 seconds. (Bottom, right) 7 seconds.

parts of the space but are only used in selected regions of the space will be pruned too early. Figure 6 shows that when there are still 36 incoming motion segments remaining, the stepping motion has been lost; the character cannot cross the blue obstacle. With 8 segments remaining, the SCC is reduced to a very small area around the pillars. Compare these results to the good coverage resulting from local pruning when only 4 incoming motion segments are maintained per grid point (Figure 5 right).

Although a two-dimensional display is used in our figures, the task domain is three-dimensional and includes character orientation. Specifically, Figures 1, 5, and 6 are projections onto the (x,z) plane of 24 slices at separate character orientations. The projection is done as a sum; variability for all orientation slices is added to produce the grey levels shown in the figures. Individual orientation slices can be examined for more information, however. Figure 7 shows two orientation directions, where the character is facing to the left and right sides of the figure. Examining these figures highlights some asymmetries in the motion. From these figures we can expect that there will be more motion choices for the character to walk clockwise through the lower left hand corner of the environment than for it to walk counterclockwise.

The embedded graph is a useful data structure for making specific queries about a character's capabilities. To evaluate a motion

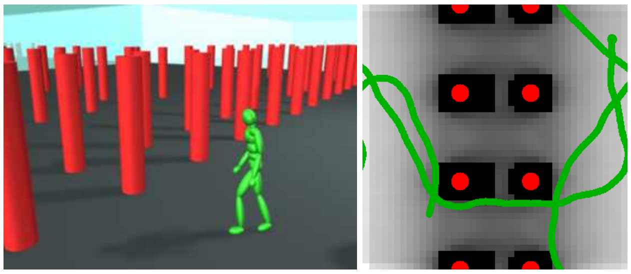

Figure 9: (Left) The character is following a random path through a motion graph that has been embedded into an environment tiled with a repeating pattern. (Right) Variablity for a single tile. The random path wraps around from left to right, bottom to top, then right to left.

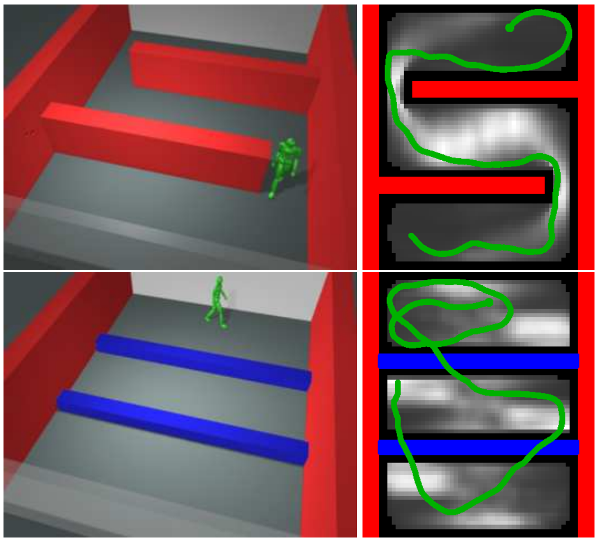

Figure 10: Additional examples. (Top) The character is following a random path through an s-shaped environment. (Bottom) The character must step over the blue obstacles to reach different parts of the space.

graph for game play in a basketball game, for example, it may be interesting to see from where the character can make a layup within three seconds. As a simple illustration of this idea, Figure 8 shows areas from which the character can enter the top region of the test environment within 2, 3, 5, and 7 seconds. Within 7 seconds the character is able to cross from the upper region into the lower region, turn around, and return to the upper region.

Large open spaces or spaces that repeat with a relatively compact pattern can easily be accommodated. Figure 9 shows coverage over a rectangle that can be tiled to cover as much space as necessary. The rectangle has the topology of a torus. Motion that sends the character off the top side of this rectangle, for example, wraps around to enter the bottom side of the rectangle. In this example, the tile size was set to 6m by 6m and grid resolution was 15cm by 15cm by 15 degrees.

Figure 10 shows variability for some additional examples. The first is a walk through an s-shaped hallway, and the second contains two obstacles over which the character must step. Note the asymmetries in the second example. The bright areas on the left hand side correspond to motions where the character is facing toward the top of the figure. The bright areas on the right hand side correspond to motions where the character is facing toward the bottom of the figure, suggesting as in Figure 7 that there may be a tendency toward clockwise motion.

6 Discussion

This is to our knowledge the first attempt to quantify the effectiveness of a motion graph data structure for a particular application. Embedding a motion graph into the task space allows us to identify weak points that arise when applying the motion graph to the given task. We showed how a local estimate of variability can be used to tune a motion graph so that its content and size are more appropriate for a given task. The embedded and tuned motion graph could be used for efficient planning of character motion at runtime.

In terms of limitations, the algorithm is best suited for tasks of low dimension, environments that can be tiled, or highly constrained environments. Low task dimension is important because the number of possible grid elements scales exponentially with task dimension. Fortunately many tasks—locomotion tasks in particular—do have this property of low dimensionality. Environments that can be tiled fairly compactly, such as open space or the environment in Figure 9, have relatively low overhead, as the graph need only be embedded into a single tile. Highly constrained environments are useful because they allow many potential links to be pruned early in the embedding process. For example, adding the step obstacle in the example shown in Figure 1 resulted in an embedded graph with 33% of the nodes and 18% of the links of the graph embedded into the same environment but without the step obstacle.

The algorithm scales linearly with the number of links in the original motion graph, due to the time required to complete the one-step unrolling process shown in Figure 3 and to find the resulting SCC. (Unrolling is the most time consuming part of the algorithm, although it would be very easy to parallelize.) Right now, we can handle motion graphs having a few hundred links for environments of the size and complexity shown in this paper before we hit memory limitations. Motion graphs of this small size may be sufficient to obtain good performance for characters engaged in game play, navigation, or other focused activity. To capture a broader spectrum of motion or many subtle variations on motion, variations to this approach will be needed. A strategy of creating separate embedded SCC's for each distinct class of behavior and then forming connections between those SCC's as in the work of Kovar and his colleagues [2002a] may be helpful to reduce the size of the motion graph that must be considered at any one time. A hierarchical approach based on clustering motion clips and operating on representative clips within each cluster similar to that of Arikan and his colleagues [Arikan et al. 2003] would also be interesting to explore for better scalability with respect to motion graph size.

The algorithms presented in this paper are appropriate for the portion of the task domain that is static. However, we note that good coverage and variability within the static portion of the environment are a necessary condition for good performance when dynamic objects are added.

Evaluation and tuning of the embedded graph is a rich domain for further investigation. Coverage and variability are convenient estimates of the character's performance because they are local; they are fast to compute and scale well with complexity of the graph. In addition, they directly provide useful information about serious problems with a data set, such as inability to achieve certain goals within a task domain. However, there are many desirable aspects of a well-tuned motion graph that local metrics cannot capture. Even if variability is good, for example, the embedded graph may contain awkwardly long paths where the character appears to take "the long way around." It also might have poor "balance" in the sense of having more left turns than right turns, for example. One area of future work is to explore better ways to estimate link utility and to perform evaluation and tuning based on more global metrics.

We would also like to consider more complex questions and task domains. Tasks such as climbing, which require dealing with more complex landscapes, should be straightforward to handle. For motions such as dance, consideration of the different motion types will be important. For example, if ballet motions are captured individually or in short sequences, will the resulting motion graph allow a full range of dance behavior on a small virtual stage?

The primary competition to our approach may be evaluation based on a controllability argument. In robotics, controllability is the ability to get from any state to any other state in a finite number of steps [Luenberger 1979]. Controllability is often evaluated based on local reasoning (e.g., the ability to move in any direction at any time). Kovar and his colleagues [2002a] illustrate something of this concept by having the character trace out complex paths on the ground plane. If this concept could be formalized, it could allow an alternative evaluation of character capability for a given motion graph data structure. An approach that relies on local analysis has the potential to provide more general results, and we are interested in exploring this option further.

One of our longer term visions is to use tools such as these to direct a motion capture session. In some motion capture scenarios, the talent is only available for a short period of time. It is important to collect all of the motion that will later be necessary. Imagine monitoring the motion database as it is being collected, evaluating the intermediate results for an intended task, and suggesting new motions that should be obtained to fill in any gaps (e.g., by extrapolating from existing motions).

Finally, we note that the evaluation and tuning algorithms described in this paper could even be applied to alternative techniques for generating motion. For example, a character controlled by a procedural algorithm could be run through its repertoire of actions, either systematically or by hand (i.e., under interactive user control). The resulting motion could be treated as motion capture data, embedded into a task domain, evaluated, and tuned to consider how the character might operate in that domain.

In summary, the techniques shown here provide a way to evaluate a character's capabilities that is more sound than the trial and error approach commonly used, and also provide additional benefits such as the ability to fine tune a data structure for desired coverage, variability, and graph size to improve runtime performance.

References

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

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2003. Motion synthesis from annotations. *ACM Transactions on Graphics 22*, 3, 402–408.

BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *Proceedings of ACM SIGGRAPH 2000*, 183–192.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2, 182–203.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*. MIT Press.

DONALD, B., XAVIER, P., CANNY, J., AND REIF, J. 1993. Kinodynamic motion planning. *Journal of the ACM 40*, 5, 1048–1066.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JAPSEN, A. 2003. Snap-together motion: Assembling run-time animations. In *Proceedings of 2003 Symposium on Interactive 3D Graphics*.

KALISIAK, M., AND VAN DE PANNE, M. 2001. A grasp-based motion planning algorithm for character animation. *Journal of Visualization and Computer Animation 12*, 3, 117–129.

KAVRAKI, L. E., AND LATOMBE, J.-C. 1998. Probabilistic roadmaps for robot path planning. In *Practical Motion Planning In Robotics: Current Approaches and Future Directions*, K. Gupta and A. del Pobil, Eds. John Wiley, 33–53.

KIM, T., PARK, S. I., AND SHIN, S. Y. 2003. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics 22*, 3, 392–401.

KOGA, Y., KONDO, K., KUFFNER, J., AND LATOMBE, J. C. 1994. Planning motions with intentions. In *Proceedings of ACM SIGGRAPH 1994*, 395–408.

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

KOVAR, L., GLEICHER, M., AND SCHREINER, J. 2002. Footskate cleanup for motion capture editing. In *Proceedings of ACM Symposium on Computer Animation 2002*.

KUFFNER, J. J., KAGAMI, S., NISHIWAKI, K., INABA, M., AND INOUE, H. 2002. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots 12*, 2, 105–118.

LATOMBE, J. C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers, Boston.

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.

LENGYEL, J., REICHERT, M., DONALD, B. R., AND GREENBERG, D. P. 1990. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics (Proceedings of ACM SIGGRAPH 90) 24*, 4, 327–335.

LI, Y., WANG, T., AND SHUM, H.-Y. 2002. Motion texture: a two-level statistical model for character motion synthesis. *ACM Transactions on Graphics 21*, 3, 465–472.

LOZANO-PÉREZ, T., AND O'DONNELL, P. A. 1991. Parallel robot motion planning. In *Proc. IEEE Intl. Conference on Robotics and Automation*.

LUENBERGER, D. G. 1979. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley & Sons.

MOLINA-TANCO, L., AND HILTON, A. 2000. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proc. IEEE Workshop on Human Motion 2000*.

PETTRÉ, J., LAUMOND, J.-P., AND SIMÉON, T. 2003. A 2-stages locomotion planner for digital actors. In *Proceedings of ACM Symposium on Computer Animation 2003*.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics 21*, 3, 501–508.

REITSMA, P. S. A., AND POLLARD, N. S. 2003. Perceptual metrics for character animation: sensitivity to errors in ballistic motion. *ACM Transactions on Graphics 22*, 3, 537–542.

ROSE, C. F., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications September/October*, 32–40.

SHILLER, Z., YAMANE, K., AND NAKAMURA, Y. 2001. Planning motion patterns of human figures using a multi-layered grid and the dynamics filter. In *Proc. IEEE Intl. Conference on Robotics and Automation*.