Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2009)
E. Grinspun and J. Hodgins (Editors)

Automatic Construction of a Minimum Size Motion Graph

Liming Zhao†, Aline Normoyle, Sanjeev Khanna and Alla Safonova

University of Pennsylvania

Abstract

Motion capture data have been used effectively in many areas of human motion synthesis. Among those, motion graph-based approaches have shown great promise for novice users due to their ability to generate long motions and the fully automatic process of motion synthesis. The performance of motion graph based approaches, however, relies heavily on selecting a good set of motions used to build the graph. This motion set needs to contain enough motions to achieve good connectivity and smooth transitions. At the same time, the motion set needs to be small for fast motion synthesis. Manually selecting a good motion set that achieves these requirements is difficult, especially given that motion capture databases are growing larger to provide a richer variety of human motions. Therefore we propose an automatic approach to select a good motion set. We cast the motion selection problem as a search for a minimum size subgraph from a large motion graph representing the motion capture database and propose an efficient algorithm, called the Iterative Sub-graph Algorithm, to find a good approximation to the optimal solution. Our approach especially benefits novice users who desire simple and fully automatic motion synthesis tools, such as motion graph-based techniques.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Computer Animation

1. Introduction

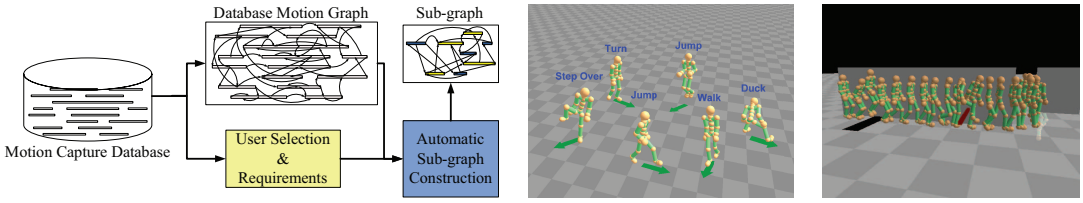
Nowadays, more and more motion capture data is becoming freely available to the public (e.g., CMU database and HDM05 [CMU, MRC*]). This data may potentially enable novice users such as children, school teachers, and small companies to easily synthesize human motions for creating simple applications. To explore this potential, there need to be fully automatic methods for human motion synthesis. Motion graphs which embed multiple behaviors into a simple graph structure have emerged as a popular data structure that can be used for automatic natural human motion synthesis. However, the performance of a motion graph is limited by the motions that the graph is built from.

Current practice for finding the motions to build a motion graph is through manual selection from a motion capture database. For example, in order to create a good quality motion graph for an interactive control application, a user selects a set of walking, running, jumping and ducking motions that are required by the application. These selected motions need to be well-connected to provide quick and smooth transitions. However, manually selecting a subset of motions from a large motion capture database to achieve this goal is an arduous process. On one hand, the subset needs to be of *the smallest possible size* in order to produce a small motion graph for efficient motion synthesis. On the other hand, the subset needs to *contain enough motion data* to preserve good quality transitions and to satisfy user requirements. As a result, the subset selection requires the user to carefully consider every relevant motion in the database, which is a highly tedious process. More importantly, estimating the effect of a particular motion on the well-connectedness of the motion graph before the graph is constructed is nearly impossible. Motivated by reducing this burden on novice users, we develop an automatic method for selecting a good but small subset of motions from a large motion capture database that can be used to build a good quality motion graph.

Our method treats the entire motion capture database as a single large motion graph and casts the motion selection problem as a search for a minimum size sub-graph from a large graph (Figure 1). We require the user to first select a small set of key motions that are required by his/her ap-

† e-mail: liming@seas.upenn.edu

SCA 2009, New Orleans, LA, August 1–2, 2009.
© 2009 ACM 978-1-60558-610-6/09/0008 $10.00



(a) (b) (c)

Figure 1: *(a) System diagram. (b) and (c) Application of our automatically computed sub-graph to interactive control applications and to off-line motion synthesis applications.*

plication domain. These motions alone often fail to generate a good motion graph that satisfies all the user requirements, because they lack quick and smooth transitions between them. Our method then automatically finds additional motions in the motion database that result in good connectivity between user selected motions, but at the same time keeps the size of the motion graph as small as possible. This step corresponds to a well-defined theoretical problem of finding a minimum size sub-graph from a very large graph. While this problem is NP-hard, we propose an efficient algorithm, called the *Iterative Sub-graph Algorithm*, that provides a good approximation to the optimal solution. Our experimental results and user study demonstrate a drastic complexity reduction in the construction of a good quality motion graph, making the process much friendlier to novice users.

2. Related Work

To our knowledge, little work has been done so far to develop an automatic method for selecting a minimum subset of motions from a large motion capture database to construct a good quality motion graph that satisfies user requirements.

Manually selecting motions for a specific task is difficult. Cooper and his colleagues solve this problem using an active learning method. Their system automatically identifies specific tasks that a motion controller performs poorly and advises a user to capture more motions to improve the performance of the controller or automatically synthesizes new motion segments [CHP07]. However, their system does not make formal guarantees on being able to perform every task from every start state. Neither does the system minimize the motion size for building the controller. Our approach guarantees to satisfy all the user requirements and minimizes the motion graph size in a fully automatic manner.

Structured graphs (often called *move trees*) [Men99, MMC01,LK05] are commonly used in industry. To construct a move tree, game designers first design all the behaviors required by the game and all the logically allowed transitions between the behaviors. Then, they carefully capture the motions according to the pre-planned behaviors so that initial motions have similar start and end postures. Finally, they hand edit the motion capture clips and choose the exact place in the motions for good transitions to appear. With the extensive amount of manual work, move trees are capable of producing motions with controlled motion quality and transition time. However, the manpower and the resources for such intensive manual design and editing are not available to the majority of animation users. Our main goal is to allow users who do not have access to these resources to easily create controllers by utilizing large amounts of motion capture data that is freely available today.

In this work we present an algorithm for automatically selecting motions to construct small size *standard* motion graphs (e.g. [LCR*02, KGP02]). We deliberately wanted to have an algorithm that generates standard graphs, because multiple algorithms have already been developed to use them for both interactive control [LCR*02, KG03] and off-line motion synthesis applications [AF02, LCR*02, LP02, AFO03, FP03, SH07]. An interesting next step would be to generalize our automatic motion selection idea to other variations of motion graphs that have been proposed over the past few years [PSS02,PSKS04,KS05,SO06,HG07,TLP07].

Zhao and Safonova [ZS08] propose a method to construct a small standard motion graph from a set of given motions to achieve both good connectivity and smooth transitions. Good connectivity allows a pose in one behavior to quickly transition to a pose in another behavior and smooth transitions guarantees good visual quality. Ikemoto and her colleagues [IAF07] propose another relevant work for creating quick transitions between all frames using clustering and classification techniques. Both algorithms start from a given set of motions and therefore do not solve the motion selection problem we address in this paper. Beaudoin and his colleagues [BvdPPC08] use a string-based clustering method to automatically organize a large motion capture database into a compact and understandable graph structure. However, the compact graph is not structured in a way to best fit for a specific user task nor does it minimize the graph size.

Our sub-graph construction algorithm is closely related to a particular form of a well-defined graph problem called the Constrained Directed Steiner Tree problem. Its goal is to find a minimum cost tree in a directed graph that connects a root vertex to a set of terminal vertices. The tree may include vertices other than the terminal vertices, known as the Steiner vertices. This problem is NP-hard and several heuristic algorithms have been developed to solve it sub-optimally [CCyC*,SKNKM06,VHG08,ALNS04]. Unfortu-

nately, most known approaches are computationally expensive with time complexity ranging from $O(k \cdot V^2)$ to $O(V^3)$, where k is the terminal size and V is the vertex size. They work sufficiently fast in the application domains such as multi-cast routing problems, where V is about 50 to 100 and k is about 50. Given the large scale of our problem, with $V > 80,000$ and $k > 500$, these algorithms fail to find a solution in a reasonable time. However, two heuristics commonly used by these algorithms are important in improving the optimality of the solution. First, it is important to connect the root vertex to the terminal vertices in an order such that early connections benefit later connections. Second it is important to identify good branch vertices. These branch vertices connect to multiple terminal vertices and therefore are the key to minimizing the total tree cost. Inspired by these heuristics, we present an efficient iterative algorithm that scales to large graphs with complexity of $O(k^2 \cdot E)$, where E is the edge size.

3. Algorithm Description

We believe that as motion capture data accumulates, it will provide a dense coverage over the behavior space and will be able to satisfy a variety of user constraints. The goal of our algorithm is to automatically select a subset of motions from a large motion capture database which results in a minimum size motion graph that satisfies user requirements. Figure 1 gives a high level overview of our approach. We first build a motion graph from a large motion capture database (Section 3.1). A user then specifies a set of key motions and a set of requirements that reflect the user's application needs (Section 3.2). We then invoke our *Iterative Sub-graph Algorithm* that finds additional motion segments in motion capture database necessary to connect well user selected motions and automatically constructs a minimum size sub-graph that satisfies user requirements (Section 3.3).

3.1. Database Motion Graph Construction

A standard motion graph is built from a large motion capture database using a common construction method [LCR*02, KGP02, ZS08]. We call this graph the *Database Motion Graph* or graph DMG for short. Each frame in the motion database is associated with a unique vertex in the graph and a directed edge exists between vertices (frames) with similarity value smaller than a user specified threshold value. To speed up the N^2 similarity comparison, where N is total frame number, we cluster the frames and only compare them within the clusters [KMN*02]. This gives us a 3 to 6 times construction time improvement.

3.2. User Input

The user input to our system is a set of key motions and a set of requirements. First, given a large motion capture database, a user selects a small set of motion segments that are required by the application, for example, some walking, turning, and jumping motion segments for an interactive control application. In this step, the user only needs to concentrate on the important aspects of the motions, such as the flight phase for a jumping behavior. The user does not need to worry about the connectivity and transition smoothness between these key motions. Our algorithm will automatically connect them using the best possible additional motions. Alternatively, since the motion capture database is annotated with behavior information, the user can just simply specify desired behavior types. The system can then automatically find motion segments with matching behavior information. The user selected key motions correspond to a set of vertices in graph DMG, which are referred as the *user selected vertices*.

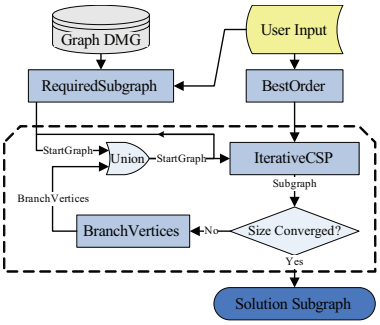


Figure 2: *Flow chart of the Iterative Sub-graph Algorithm.*

Next, the user specifies requirements for his/her application. For example, interactive control applications require graphs with fast transitions between frames to guarantee quick response to user input. One straightforward way to generate such a graph is to limit the transition time between all pairs of user selected vertices to at most Γ frames defined by the user. Sometimes, even the shortest transition in the original database exceeds Γ frames. In that case, we use the actual minimum length as the constraint instead of Γ. For easy understanding of our algorithm, we explain it based on this user requirement. In Section 3.4, we extend our algorithm to handle more complex user requirements, such as constraints on local maneuverability measure [RP07] and constraints on transition quality.

3.3. Iterative Sub-graph Algorithm (ISA)

Figure 2 gives an overview of our Iterative Sub-graph Algorithm (ISA). ISA computes a *Subgraph* that satisfies user requirements using our *IterativeCSP* algorithm (Section 3.3.4). At each iteration, the *IterativeCSP* connects the user selected vertex pairs with constrained shortest paths from graph DMG. The optimality of the solution depends on the order in which *IterativeCSP* processes these vertex pairs. We use a predetermined processing order (the *Be-*

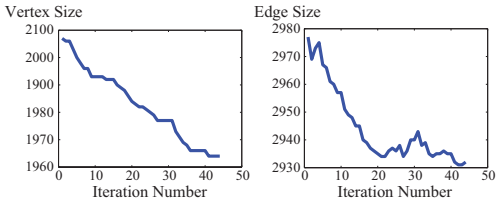


Figure 3: *Convergence Plot of ISA.*

stOrder block) to achieve good solution optimality and computational efficiency. Section 3.3.1 discusses the order we choose. The optimality of the solution also depends on the graph *IterativeCSP* starts with (*StartGraph*), which is improved at each iteration. Initially, the *StartGraph* is set to the the *RequiredSubgraph*, which contains all the vertices that must be included in the solution in order to satisfy all the user constraints. Section 3.3.2 describes the computation of the *RequiredSubgraph*. The *RequiredSubgraph* itself does not satisfy all the user constraints because many of the user selected vertices are either disconnected or connected by a path longer than Γ frames. *IterativeCSP* starts from this graph and computes a *Subgraph* that satisfies user requirements. At the end of each iteration, we evaluate the resulting *Subgraph* to identify vertices with high branching factors, called *BranchVertices* (the *BranchVertices* block). Section 3.3.3 explains the identification of *BranchVertices*. The *BranchVertices* and the *StartGraph* from current iteration comprise a better *StartGraph* for minimizing the *Subgraph* size in the next iteration (the *Union* operation). ISA terminates when the *Subgraph* size converges to a minimum number. Figure 3 shows a graph size convergence example.

3.3.1. Computing BestOrder

The optimality of the solution depends on the order in which *IterativeCSP* processes user selected vertex pairs. According to the Steiner problem heuristics, a good order ensures that earlier connections benefit later connections. A number of metrics have been developed for choosing the order [CCyC*, SKNKM06, VHG08, ALNS04] at run time. However, these metrics are computationally expensive. Given the large scale of our problem, we instead choose a predetermined order to approximate this heuristic.

We base our order on the length of the shortest paths between the user selected vertices in the original motion database. We have experimented with descending order, ascending order, random order and sequential order according to the vertex index. Experiments show that the descending order (pairs with the longer length to be processed first) consistently performs the best. Intuitively, by finding a constrained shortest path connecting the user selected vertex pairs that are far apart, the search automatically finds constrained paths between the intermediate user selected vertices, avoiding the necessity for computing these vertex pairs with shorter transition lengths later. Therefore, the descending processing order approximates the processing order heuristic of ensuring that earlier connections benefit later connections. Table 1a shows the effect of different processing orders. The "Time" column shows the computation time per iteration of ISA. The descending order consistently produces smaller sub-graphs in less time. The exact database and user input used are explained in Section 4.

		Vertex	Edge	Time
a	**Descending**	2,007	2,977	17 sec
	Ascending	2,049	3,182	5 min
	Random	2,011	3,057	3 min
	Sequential	2,038	3,140	4 min
b	**No ReqSub**	2,249	3,378	N/A
	ReqSub	1,961	2,932	N/A

Table 1: *Effect of the processing order (a) and the RequiredSubgraph (b)*

3.3.2. Computing RequiredSubgraph

Some vertices and edges *must* appear in the solution subgraph, such as the user selected vertices and edges between them. In addition, some of the vertices in graph DMG, if absent, will cause graph DMG to violate user constraints. Because these vertices are guaranteed to be part of the solution and in order to achieve more optimal solution these vertices should be added to *StartGraph* from the beginning. To find these vertices, we remove a candidate vertex from graph DMG and compute shortest paths between all user selected vertices. If any path is longer than Γ, this candidate vertex is declared to be part of the RequiredSubgraph. In practice, to save computation time, we don't test every vertex in graph DMG. Instead, we first compute a sub-graph using one iteration of ISA with *StartGraph* initialized just with the user selected vertices and edges between them. Only these vertices that belong to this sub-graph could be part of the RequiredSubgraph and therefore need to be tested. Table 1b shows the sub-graph computed by ISA when starting without RequiredSubgraph ("No ReqSub" row) and starting with RequiredSubgraph ("ReqSub" row). The one that starts from the RequiredSubgraph achieves smaller graph size. The exact database and user input used are explained in Section 4.

3.3.3. Computing BranchVertices

BranchVertices are vertices that contribute to multiple shortest paths between user selected vertices. Because these vertices lie on many shortest paths and the goal of our algorithm is to minimize the sub-graph size, we get a smaller sub-graph if we include these vertices to the *StartGraph* as early as possible. To identify good BranchVertices from a given sub-graph, we run Dijkstra's shortest path search between all the user selected vertices. For every vertex along each shortest path, we increase its counter by 1. Therefore, the count reflects how many shortest paths contain that vertex. Finally, vertices with higher counts that are not already in the StartGraph are declared as good BranchVertices. Figure 3 illustrates the effect of BranchVertices on minimizing the sub-graph size as the algorithm iterates.

3.3.4. Iterative Constrained Shortest Path Search (IterativeCSP)

Algorithm 1: IterativeCSP

Data: *DMG*: the database motion graph, Γ: the transition time upper bound.
Input: *StartGraph*: the starting graph, *BestOrder*: the predetermined processing order
Output: *Subgraph*: the solution sub-graph

```
1 Subgraph ← StartGraph;
2 UpdateGraph(Subgraph, DMG);
3 {(v_s^1, v_t^1), ... , (v_s^K, v_t^K)}← BestOrder;
4 for i = 1 to K do
5     p ←ShortestPath(v_s^i, v_t^i, SubGraph);
6     if length(p)> Γ then
7         p ←CSP(v_s^i, v_t^i, DMG, Γ);
8         Subgraph ← Subgraph ⋃ p;
9         UpdateGraph(Subgraph, DMG);
```

The IterativeCSP algorithm (Algorithm 1) takes as input a StartGraph and the BestOrder to process the user selected vertex pairs. First, it initializes a Subgraph to be the StartGraph. Second, it calls the UpdateGraph(Subgraph, DMG) function to update the cost of all edges in graph DMG such that the cost is set to 0 if the edge is incident to any vertex in the current Subgraph, otherwise it is set to 1. This way, the total cost of a path in graph DMG is the number of vertices outside the current Subgraph, which is what our algorithm tries to minimize. Assuming we have k user selected vertices, the number of user selected vertex pairs are $K = k^2$. IterativeCSP arranges them $((v_s^1, v_t^1), \ldots, (v_s^K, v_t^K))$ according to the BestOrder and iterates through each of them. If the current Subgraph contains a shortest path that satisfies the transition time constraint Γ, it moves on to the next vertex pair. Otherwise, it invokes a Constrained Shortest Path Search (CSP) to connect the two vertices. This path is added to the current Subgraph and graph DMG is updated accordingly by calling the UpdateGraph(Subgraph, DMG) function.

Constrained Shortest Path Search (CSP): Since the cost of each edge in graph DMG is set to 0 if the edge is incident to any vertex in the current Subgraph and otherwise the cost is set to 1, Constrained Shortest Path Search (CSP) finds a path that introduces minimum number of new vertices to the current Subgraph with length no larger than Γ. This particular CSP problem can be solved efficiently using a standard Dynamic Programming algorithm with the following cost-to-go functions:

$$J(v,1) = \begin{cases} c(v_s,v) & \text{if } (v_s,v) \in E \\ \infty & \text{otherwise} \end{cases} \quad (1a)$$

$$J(v,l) = \min\{\min_{(u,v)\in E} J(u,l-1) + c(u,v),\ J(v,l-1)\} \quad (1b)$$

where v_s is the source vertex, $c(u,v)$ is the edge cost in graph $DMG = (V,E)$, $J(v,l)$ is the minimum cost to get to v from v_s with length $\leq l$, and $v \in V$, $l = 1,2,\cdots,\Gamma$.

3.3.5. Complexity Analysis

Finally, we analyze the complexity of our algorithm. The complexity of CSP is $O(\Gamma \cdot E)$, where E is edge size of graph DMG. Given that Γ is often a small constant, the complexity of CSP can be considered as $O(E)$. Therefore, the complexity of IterativeCSP is $O(k^2 \cdot E)$ in the worst case, where k is the number of user selected vertices. We run k Dijkstra's shortest path search on the solution sub-graph to identify good BranchVertices. Note that the sub-graph size is usually much smaller than graph DMG. Nevertheless, the complexity for identifying BranchVertices is $O(k(E + V\log V))$ in the worst case. Since edge size E dominates in graph DMG, the complexity of ISA is $O(k^2 \cdot E)$ per iteration. Usually, the computation converges within 50 to 60 iterations.

3.4. Improvements and Extensions

In this section, we discuss improvements to scale our algorithm to large motion capture databases and extend our algorithm to handle more complex user requirements.

3.4.1. Scaling to large motion capture databases

Motion capture databases accumulate a variety of motions and provide a good basis for new motion synthesis. However, given that a particular application only needs a small set of behaviors such as walking behaviors, it is questionable whether we need to compute the sub-graph from the full database that includes jumping, ducking and dancing behaviors. Therefore, we propose restricting our ISA to only consider vertices and edges from a much smaller representative graph, called graph *CORE*. Graph CORE is a union of shortest paths between all the user selected vertices in graph DMG. Our numerous experiments show that graph CORE consistently contains over 98% of the vertices that comprise the sub-graph computed by our algorithm starting from graph DMG. Therefore, graph CORE provides almost as good a starting point as graph DMG.

3.4.2. Extending to other user requirements

We have been describing our algorithm based on the frame to frame transition time requirement, which we call **Requirement 1**. Now we extend our algorithm to handle more complex requirements.

Requirement 2: A direct extension to Requirement 1 is to require that the sub-graph preserve the minimum transition time between all the frames in the database. To satisfy this Requirement, Γ is replaced by a function that returns the minimum transition time in graph DMG between user selected vertex pairs. The rest of the algorithm stays the same.

Requirement 3: Reitsma and Pollard [RP07] proposed the local maneuverability (LM) measurement, which computes the average minimum time to transition between clips and behaviors. We adapt our ISA to handle clip to clip LM measure constraints. The user specifies an upper bound Γ for

clip to clip LM. In IterativeCSP, instead of processing every vertex pair, we run a single-source multiple-target CSP search for every clip pair to find a constrained least cost path from the last frame of one clip to a set of frames at the beginning of another clip. The rest of the algorithm stays the same. This requirement is often useful for applications that emphasize motion realism over immediate transitions. These applications often require the key motions to complete before transitions to other behaviors.

Requirement 4: We assume that as motion capture data accumulates over years, it will provide a denser coverage over the behavior space. Therefore, graph DMG can be constructed at a low similarity threshold to contain only smooth transitions. As a result, all transitions are treated as having equal motion quality. In reality, such dense coverage is often not feasible and graph DMG needs to be constructed at a higher similarity threshold to connect multiple behaviors. Therefore, graph DMG will contain different quality transitions. We now show how to set up our algorithm to handle constraints on transition quality (not just transition length), which is defined by the total cost of the corresponding path. Instead of encoding the edges in graph DMG with constant cost 1 to constrain transition length, we now set the edge cost as follows:

$$c = \alpha c_t + \gamma \tag{2}$$

where c is the edge cost, $c_t \in [0,1]$ is the transition cost computed from the similarity metric to reflect the quality of the transition, with 0 being the best and 1 being the worst, γ is a small constant representing the cost of 1 frame of transition time, and α balances between transition quality and transition time. Changes to the current algorithms includes the following. The BestOrder will be computed based on the transition quality instead of the transition length. UpdateGraph(Subgraph, DMG) sets the edge cost of graph DMG to αc_t if it is incident to a vertex in the current Subgraph; otherwise it is set to $\alpha c_t + \gamma$. This way, the minimum cost path is a smooth motion with a minimum number of new frames (vertices). The CSP algorithm uses transition quality as the constraint instead of transition length and we discretize the transition quality to fit into the Dynamic Programming framework. The rest of the algorithm stays the same.

4. Experimental Analysis

In this section, we analyze the performance of our Iterative Sub-graph Algorithm. For most of the experiments presented in this section, the user selection contains 504 frames from 13 motion segments. These motion segments are the key portions of some walking, standing, turning, jumping, ducking and stepping over behaviors, that targeted towards interactive control applications. Requirement 1 with $\Gamma = 40$ frames is used for most of the experiments to analyze the size and the connectivity of the generated motion graph. Smaller motion graphs with quicker transitions are especially useful for interactive control applications, as observed by a number of researchers [IAF07, MP07, TLP07, ZS08].

	Vertex	Edge	Time
Graph DMG	86,647	613,838	25 minutes
Graph CORE	12,040	36,583	54 seconds
Subgraph (DMG)	1,961	2,932	13 minutes
Subgraph (CORE)	2,037	2,975	7 minutes
RequiredSubgraph	1,531	491	N/A

Table 2: *Graph Size*

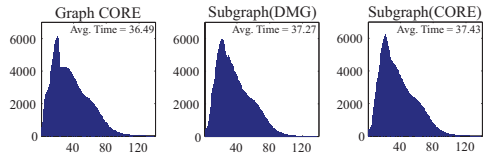


Figure 4: *Transition Time Histogram Comparison. The X axis is the transition time in terms of number of frames. The Y axis is how many transitions are of a particular length.*

Scalability: The motion capture database we used contains 89 motions with a total of 86,987 frames sampled at 30Hz. The database contains a variety of behaviors as well as natural transitions between them that occurred during the motion capture sessions. The corresponding database motion graph (graph DMG) contains 86,674 vertices and 613,838 edges, which covers over 99% of the entire database.

We use the boost graph library [BGL02] to efficiently represent each vertex with a unique integer and each edge with a real value indicating the transition cost. We store postures and their relative root transformations on the hard disk and only use them for similarity comparison during the motion graph construction and for the final motion generation. The run time memory cost is about 80MB. The construction time for this database motion graph is 25 minutes using k-means clustering. Our $O(k^2 \cdot E)$ ISA is computationally efficient with computation time ranging from 10 to 15 minutes depending on different user requirements.

To scale to even larger databases, we can resort to external memory for storage and external memory algorithms for graph search [Vit99] to first compute graph CORE. Then, as discussed in Section 3.4, the sub-graph computation can be perform from graph CORE with little compromise to the optimality of the solution.

Graph size: We analyze the performance of our algorithm through graph size comparison. We compare the size of the sub-graph computed by ISA starting from graph DMG and starting from graph CORE to the size of graph DMG, graph CORE and the RequiredSubgraph. Table 2 shows the vertex size (the "Vertex" column) and the edge size (the "Edge" column) and computation time (the "Time" column) comparison. Our sub-graph computed starting from graph DMG satisfies Requirement 1 with a very small graph size, 2% of the database size and 16% of graph CORE size (graph CORE presents a trivial solution to satisfy Requirement 1 but with a

	Vertex	Edge
Graph DMG	86,647	613,838
Graph CORE	12,040	36,583
Requirement 1	1,961	2,932
Requirement 2	4,200	10,359
Requirement 3	861	996
Requirement 4	2,217	2,810

Table 3: *Sub-graphs for All Requirements.*

large graph size). It only increases the RequiredSubgraph by 28%. At the same time, the sub-graph computed from graph CORE is only 4% larger than the sub-graph computed from graph DMG using only about half of the computation time. In conclusion, our ISA achieves a good approximation to the optimal solution and graph CORE allows us to scale to larger databases with little compromise to the solution optimality.

Graph quality: We analyze the graph quality by comparing the histograms of the minimum transition time between the user selected vertices for graph CORE, the sub-graph computed from graph DMG (Subgraph(DMG)) and the sub-graph computed from graph CORE (Subgraph(CORE)). Table 2 shows their graph sizes. Figure 4 shows the transition time histogram. Graph CORE contains the best possible paths and its average minimum transition time (Avg. Time) is 36.49 frames. Subgraph(DMG) lengthens some shorter (length ≤ 40) transitions to share paths in order to achieve a smaller graph size. In conclusion, Subgraph(DMG) and Subgraph(CORE) have very similar length distributions to the distribution for graph CORE (which is the best). Therefore, Subgraph(DMG) and Subgraph(CORE) made little compromise in transition time distribution (only about 1 frame increase to the average transition time comparing to graph CORE) to achieve a much smaller graph size, making them well suited for interactive control applications.

Constructing sub-graphs for other requirements: Table 3 shows the sub-graphs constructed for all the requirements listed in Section 3.4. The first three rows show the graph size of graph DMG, graph CORE and the sub-graph for Requirement 1. The fourth row shows the sub-graph size for Requirement 2, computing a sub-graph that preserves the best transition lengths. This is a more difficult constraint. It results in a larger sub-graph size, but the sub-graph size is still significantly smaller than graph DMG and graph CORE. The fifth row shows the sub-graph size for Requirement 3 with the clip to clip LM limit set to 15 frames. The clip to clip LM is a relative loose constraint between frame sets and results in a much smaller sub-graph size. The sixth row shows the sub-graph size for Requirement 4 on transition quality with the upper bound $\Gamma = 60$, $\alpha = 10.0$ and $\gamma = 1.0$ in Equation 2. $\Gamma = 60$ roughly corresponds to a 50-frame smooth transition constraint, definitely not exceeding 60 frames. In all cases, our ISA generates sub-graphs that satisfy user requirements with very small graph sizes.

We have also tested on some other databases of different size and with different user selections. Our algorithm consistently produces sub-graphs that satisfy user requirements with very small graph sizes, usually less than 3% of the database size and less than 20% of the graph CORE size.

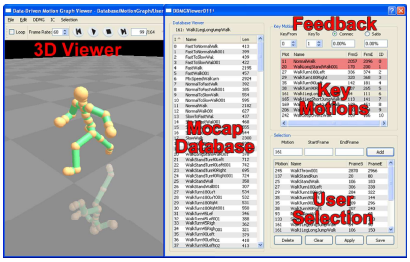


Figure 5: *The interface for manually constructing a motion graph. The left side is a 3D viewer. The middle is a motion capture database explorer and the right side is the manual motion selection panel.*

5. User Study

Our user study simulates how novice users and animation researchers usually create motion graphs by manually picking motions from large motion capture databases (such as the CMU database). The study confirms that manually selecting a set of motions from a large motion capture database that would result in a good quality motion graph and satisfy some given requirements is difficult.

We performed the user study with 8 subjects. Among them, 3 are experienced motion graph users while the other 5 are novice animation users. All subjects were presented with the motion capture database and a set of key motions described in Section 4. Through a motion graph construction interface shown in Figure 5, they browsed the motion capture database to find additional motion segments for constructing a motion graph with good connectivity and fast transitions (a 50-frame transition time constraint was the goal). The subjects were given two tasks of manually constructing motion graphs. In the first task, they were asked to select candidate motion segments and make changes as they like and then create the motion graphs to evaluate graph *connectivity* (how many key motions are connected to each other) and constraint *satisfaction* (how many transitions between the frames from the key motions are of length ≤ 50). This task simulates a normal manual motion graph construction process. In the second task, they were given constant feedback on *connectivity* and *satisfaction* of the motion graphs built from their currently selected motions to help them improve the selection. At a certain point, it became extremely difficult for the subjects to improve the selection further and they stopped the test.

Table 4 shows the average performance of novice users (Novice) vs. experienced users (Exp.) for the two tasks. We also compare the results to a sub-graph constructed under the same problem setting using our ISA (Our Alg.). The first column gives the time spent on constructing the motion

		T.(m)	Vertex	Edge	C	S	Ave. T
1	Novice	28	876	1,122	42%	24%	N/A
	Exp.	50	5,156	13,559	84%	58%	N/A
2	Novice	69	7,948	24,090	59%	30%	N/A
	Exp.	82	16,104	58,699	100%	66%	50
Our Alg.		11	2,217	2,810	100%	100%	44

Table 4: *User Study*

graph manually and the time to compute the sub-graph automatically using our ISA. The time is measured in minutes. The second and third columns show the corresponding graph sizes. The fourth and fifth columns show the graph connectivity (C) and the constraint satisfaction (S) respectively. The last column shows the average transition time between the frames from the key motions measured in number of frames. The system feedback helped the graph construction process. Only experienced users were able to create a fully connected graph with the feedback (task 2). They mentioned that the key strategy was to select more and longer motion segments - the redundancy helped. However, the average satisfaction rate is only 66%. All of the subjects reported that it is very difficult to predict which motion segment would connect two motion clips smoothly and quickly. In addition, it is impossible to locate such a segment in a large database. Our fully automatic sub-graph algorithm outperforms manual motion graph construction in all measures.

6. Application to Motion Synthesis

Application to interactive control: In this section, we demonstrate the applicability of our automatically computed motion graph (*sub-graph*) to interactive control applications. We use value iteration methods to learn good control policies following the reinforcement learning techniques commonly used by several other researchers [LL04,MP07,TLP07]. The accompanying video demonstrates both good motion quality and quick responsiveness to user control. We compare the control policies learned from our sub-graph to those learned from the best motion graph created by an expert during our user study. Our sub-graph is much smaller in size, but achieves comparable motion quality and motion responsiveness. The motion graph created by the expert contains 10,404 vertices and 33,849 edges and requires a 146MB run-time policy table from 3 hours of training. Our sub-graph contains 2,217 vertices and 2,810 edges and requires a 31MB run-time policy table from 16 minutes of training. We also performed the same comparison to the best motion graph created by a novice user. The accompanied video shows the lack of responsiveness and inability to transition to the ducking behavior by the control policy learned from this manually constructed motion graph.

Application to off-line search: Our sub-graphs are also well-suited for off-line motion synthesis. Using a publicly-available SBPL graph search library [Lik], we generate motions that navigate a character from point A to point B in an environment while avoiding different obstacles, such as walls, holes and bars. As the search progresses, each vertex (frame) in the motion graph is augmented with the global position and orientation of the root in the environment in order to satisfy the position constraint on the root of the character. As a result of the small size of our sub-graph, the search for a 12-second long motion shown in the accompanying video took only a few seconds to compute. The quick transitions provided by our sub-graph allow the search to find motions with efficient movements. The accompanying video shows the generated motions. The same search on the best motion graph constructed by an expert did not converge to a feasible solution because the large graph size caused the search to run out of memory. The same search on the best motion graph constructed by a novice user failed to find a feasible solution, because the graph lacked good connectivity between all the behaviors.

7. Conclusions and Discussion

Large motion capture databases and motion graph techniques enable both novice users and animation researchers to easily create good quality controllers for their animation applications. However, as shown in our user study, finding a good set of motions that results in a good quality motion graph is difficult. We present an algorithm for automatically selecting motions from a large database that would result in a good quality motion graph. Users control the content and the quality of the resulting motion graph by selecting key motion segments and by specifying requirements that fit their application needs. While this problem is NP-hard, our efficient Iterative Sub-graph Algorithm provides a good approximation to the optimal solution and scales to large motion capture databases. We believe that this paper is a necessary step towards making motion graphs a practical tool for a larger spectrum of animation users, in particular novice users such as children, school teachers, small companies, etc.

The quality of the sub-graph computed by our algorithm relies on the quality of the motion capture database. Even after years of accumulation, the database still lacks good motion quality and connectivity between certain behaviors. Low quality transitions are required to include these behaviors to graph DMG, and they decrease the motion quality of the generated sub-graphs. We address the motion quality issue by setting the constraints on transition quality (Requirement 4). However, transition quality is a combination of both transition smoothness and transition length. The constraint on transition quality Γ is not as intuitive to specify as the length constraints for other requirements. In the future, we would like to investigate the method proposed by Zhao and Safonova [ZS08] to improve the motion quality and connectivity of the motion capture database by introducing a minimum amount of interpolation to only include smooth transitions in graph DMG.

We deliberately targeted our algorithm towards *standard* motion graphs because of their simple graph structure and

widespread support by many existing motion synthesis algorithms. However, automatic motion selection idea will be just as beneficial to other variations of motion graphs (such as structured motion graphs with interpolations) that have been proposed over the past few years [PSS02, PSKS04, KS05, SO06, HG07, TLP07]. In the future we plan to generalize our ideas to these graphs.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Trans. on Graphics 21*, 3 (2002).

[AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *ACM Trans. on Graphics 22*, 3 (2003).

[ALNS04] AGGARWAL S., LIMAYE M., NETRAVALI A., SABNANI K.: Constrained diameter steiner trees for multicast conferences in overlay networks. *QSHINE 2004* (2004), 262–271.

[BGL02] *The boost graph library: user guide and reference manual*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[BvdPPC08] BEAUDOIN P., VAN DE PANNE M., POULIN P., COROS S.: Motion-motif graphs. In *Symposium on Computer Animation 2008* (July 2008).

[CCyC*] CHARIKAR M., CHEKURI C., YAT CHEUNG T., DAI Z., GOEL A., GUHA S., LI M.: Approximation algorithms for directed steiner problems.

[CHP07] COOPER S., HERTZMANN A., POPOVIĆ Z.: Active learning for real-time motion controllers. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers* (New York, NY, USA, 2007), ACM, p. 5.

[CMU] Motion capture database at carnegie mellon university.

[FP03] FANG A. C., POLLARD N. S.: Efficient synthesis of physically valid human motion. *ACM Trans. on Graphics 22*, 3 (2003), 417–426.

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. In *ACM Symposium on Interactive 3D Graphics* (2007).

[IAF07] IKEMOTO L., ARIKAN O., FORSYTH D.: Quick transitions with cached multi-way blends. In *ACM Symposium on Interactive 3D Graphics* (2007), pp. 145–151.

[KG03] KOVAR L., GLEICHER M.: Flexible automatic motion blending with registration curves. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (Aug. 2003), pp. 214–224.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. on Graphics 21*, 3 (2002), 473–482.

[KMN*02] KANUNGO T., MOUNT D. M., NETANYAHU N. S., PIATKO C. D., SILVERMAN R., WU A. Y.: An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Trans. Pattern Anal. Mach. Intell. 24*, 7 (2002), 881–892.

[KS05] KWON T., SHIN S. Y.: Motion modeling for on-line locomotion synthesis. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (July 2005), pp. 29–38.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Trans. on Graphics 21*, 3 (2002), 491–500.

[Lik] LIKHACHEV M.: *SBPL graph search library*. http://www.cis.upenn.edu/ maximl/software.html.

[LK05] LAU M., KUFFNER J. J.: Behavior planning for character animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Aug. 2005), pp. 271–280.

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (2004), pp. 79–87.

[LP02] LIU C. K., POPOVIĆ Z.: Synthesis of complex dynamic character motion from simple animations. *ACM Trans. on Graphics 21*, 3 (2002), 408–416.

[Men99] MENACHE A.: *Understanding Motion Capture for Computer Animation and Video Games*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[MMC01] MARK MIZUGUCHI J. B., CALVERT T.: Data driven motion transitions for interactive games. In *Eurographics 2001 Short Presentations* (2001).

[MP07] MCCANN J., POLLARD N. S.: Responsive characters from motion fragments. *ACM Trans. on Graphics (SIGGRAPH 2007)* (2007).

[MRC*] MÜLLER M., RÖDER T., CLAUSEN M., EBERHARDT B., KRÜGER B., WEBER A.: *Documentation Mocap Database HDM05*. Tech. Rep. CG-2007-2.

[PSKS04] PARK S. I., SHIN H. J., KIM T. H., SHIN S. Y.: On-line motion blending for real-time locomotion generation. *Computer Animation and Virtual Worlds 15*, 3-4 (2004), 125–138.

[PSS02] PARK S. I., SHIN H. J., SHIN S. Y.: On-line locomotion generation based on motion blending. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation* (July 2002).

[RP07] REITSMA P. S. A., POLLARD N. S.: Evaluating motion graphs for character animation. *ACM Trans. Graph.* (2007), 18.

[SH07] SAFONOVA A., HODGINS J. K.: Construction and optimal search of interpolated motion graphs. In *ACM Trans. Graph.* (2007), p. 106.

[SKNKM06] SKORIN-KAPOV, NINA, KOS, MLADEN: A grasp heuristic for the delay-constrained multicast routing problem. *Telecommunication Systems 32*, 1 (May 2006), 55–69.

[SO06] SHIN H. J., OH H. S.: Fat graphs: Constructing an interactive character with continuous controls.

[TLP07] TREUILLE A., LEE Y., POPOVIĆ Z.: Near-optimal character animation with continuous control. *ACM Trans. Graph.* (2007), 7.

[VHG08] VIK K.-H., HALVORSEN P., GRIWODZ C.: Evaluating steiner-tree heuristics and diameter variations for application layer multicast. *Comput. Netw. 52*, 15 (2008), 2872–2893.

[Vit99] VITTER J. S.: External memory algorithms and data structures. 1–38.

[ZS08] ZHAO L., SAFONOVA A.: Achieving good connectivity in motion graphs. In *Proceedings of the 2008 ACM/Eurographics Symposium on Computer Animation* (July 2008), pp. 127–136.