# Construction and optimal search of interpolated motion graphs

Alla Safonova Jessica K. Hodgins
School of Computer Science
Carnegie Mellon University


Figure 1: Optimal and sub-optimal solutions for walking a given distance (left) and for picking up an object (right).

## Abstract

Many compelling applications would become feasible if novice users had the ability to synthesize high quality human motion based only on a simple sketch and a few easily specified constraints. We approach this problem by representing the desired motion as an interpolation of two time-scaled paths through a motion graph. The graph is constructed to support interpolation and pruned for efficient search. We use an anytime version of $A^*$ search to find a globally optimal solution in this graph that satisfies the user's specification. Our approach retains the natural transitions of motion graphs and the ability to synthesize physically realistic variations provided by interpolation. We demonstrate the power of this approach by synthesizing optimal or near optimal motions that include a variety of behaviors in a single motion.

**CR Categories:** I.3.6, I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

**Keywords:** motion capture, motion graph, motion planning, human animation, motion interpolation

## 1 Introduction

The ability to construct animations of human characters easily and without significant training would enable many novel and compelling applications. Children could animate stories, novice users could author effective training scenarios, and game players could create a rich set of character motions. With these applications in mind, we have focused on techniques that require users to provide only a small amount of information about a desired motion. The user provides an approximate sketch of the path of the character on the ground plane and a set of constraints (Figure 2). Given this simple description of the desired motion, our discrete optimization approach can quickly create a natural-looking motion that matches the user's specification.

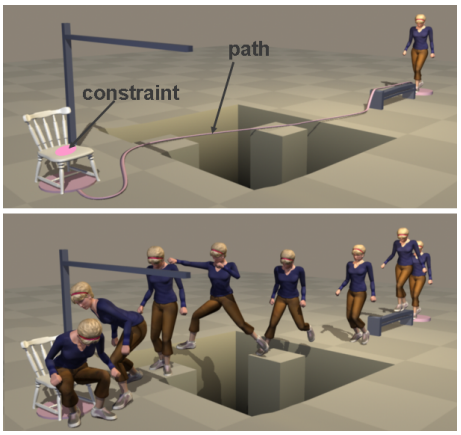



Figure 2: (Top) Rough sketch of the desired path and a user constraint requiring the character to sit on the chair at the end of the path. (Bottom) Synthesized motion.

The key insight behind our approach is that we represent the motion as an interpolation of two time-scaled paths through a motion graph. The strength of this representation is that it allows the adaptation of existing motions through interpolation while also retaining the natural transitions present in a motion graph. We allow interpolation only of segments with matching contact patterns and therefore the resulting motion is often close to physically correct [Safonova and Hodgins 2005]. This representation creates a search space that is far smaller than the full space created by the 50 degrees of freedom of a human character because it contains only natural poses and velocities from the original motions and the interpolation of segments with matching contact patterns.

We can search our representation for optimal or near-optimal solutions using an anytime version of $A^*$ [Likhachev et al. 2003]. To make a globally optimal search possible, we developed two techniques that significantly decrease the number of states that the search will have to visit. The first technique compresses the motion graph into a *practically equivalent* but much smaller graph by removing states and transitions that would not be part of an opti-

{alla, jkh}@cs.cmu.edu

mal solution or are redundant. The second technique computes an informative heuristic function that guides the search toward states that are more likely to appear in an optimal solution. The combination of these two techniques makes it possible to find optimal or close-to-optimal solutions for 15 second motion with a few minutes of computation.

The user can specify motion by sketching the path of the character through the environment and by specifying constraints. The system can only support user constraints at the transition between contact phases (for example, the transition from standing to standing while holding a cup). This restriction is less of a problem than it might appear because animated characters tend to act on their environment, and user constraints, therefore, often create contact changes (footsteps, picking up an object, sitting on a chair, for example).

We demonstrate the power of this approach with a number of examples generated from a database of 6-7 minutes of motion. The examples are lengthy and include such behaviors as walking, ducking, walking along a beam, jumping, picking up an object, and sitting down. These examples demonstrate that we have retained the key advantages of motion graphs: long motions, a variety of behaviors, and natural transitions. We demonstrate the value of interpolation through a set of examples that satisfy constraints that could not be met with the original motion graph. The value of an optimal solution is shown through examples such as those in Figure 1 where the greedy solution is less natural than the optimal solution. The optimality of our solution avoids the dithering or unnatural motions that are sometimes found when motion graphs are searched with greedy or locally optimal algorithms.

In the next section we review related work. We then define the optimization problem and explain how the graph is constructed, compressed, and searched efficiently. We conclude with experimental results in Section 7 and a discussion of the advantages and limitations of our approach in Section 8.

## 2 Background

Continuous optimization, introduced to the graphics community by Witkin and Kass [1988], is a common technique for finding a motion when only a rough sketch is provided. These techniques rely on physical laws to constrain the search and produce natural-looking motion. Continuous optimization has been shown to work well when a good initial guess is provided and for synthesizing relatively short, single behavior motions, such as jumps and runs (see for example [Fang and Pollard 2003; Safonova et al. 2004; Sulejmanpašić and Popović 2005]). In contrast to continuous optimization, the discrete optimization approach explored in this paper can handle longer motions that consist of multiple behaviors and does not require an initial guess. However, it is restricted to resequencing and interpolation of prerecorded motions and cannot generate entirely novel motions as continuous optimization can.

Interpolation is a simple, yet powerful technique for generating motions that are variations of motions in a database [Perlin 1995; Guo and Roberge 1996; Wiley and Hahn 1997]. Rose and his colleagues [1998] implemented a system that allowed interactive control of a character driven by interpolated motion. Kovar and Gleicher [2003; 2004] created a technique for identifying motions that could be interpolated and for automatically registering those motions. Abe and his colleagues [2004] used optimization to synthesize a family of highly dynamic motions from a motion capture sequence and then interpolated those motions to create others. Safonova and Hodgins [2005] analyzed interpolated motions for physical correctness and showed that interpolation produces motions that are close to physically correct in many cases. Our discrete optimization approach also relies on interpolation to synthesize variations of existing motions, but does not require that the example motions be aligned in advance. Instead, the subsequences that are interpolated are selected automatically during the optimization process.

Motion graphs and related approaches can be categorized into *on-line* approaches where the motion is generated in response to user input (from a joystick, for example) [Lee et al. 2002] and *off-line* approaches where the full motion specification is known in advance [Arikan and Forsyth 2002; Kovar et al. 2002]. On-line approaches can perform only local search because new input is continuously arriving. Off-line approaches, on the other hand, can find a high quality solution that minimizes an objective function such as energy. Our work falls into the category of off-line techniques.

A number of algorithms have been developed to search a motion graph in an off-line fashion [Arikan and Forsyth 2002; Kovar et al. 2002; Pullen and Bregler 2002; Arikan et al. 2003; Choi et al. 2003; Sung et al. 2005; Li et al. 2002; Srinivasan et al. 2005]. These techniques either use a global but sub-optimal search or concatenate a series of short motion segments each found by a separate search. To find an optimal solution efficiently, Lau and Kuffner [2005] manually created a behavior-based motion graph with a very small number of nodes. In later work, they precomputed search trees from the graph and used them for faster but not globally optimal search [Lau and Kuffner 2006]. Lee and Lee [2004] precomputed policies that indicate how the avatar should move for each possible control input and avatar state. Their approach allows interactive control with minimal run-time cost for a restricted set of control inputs.

A number of on-line approaches have been created in the past few years that combine motion graphs and interpolation techniques. Park and his colleagues [2002; 2004] manually preprocess motion into short segments. Segments with similar structure are then arranged into nodes in a graph and blended so that local search can be used to generate locomotion in real-time. The graph construction was later automated for locomotion by Kwon and Shin [2005]. Shin and Oh [2006] extended these techniques to include additional behaviors. They use a method similar to the one proposed by Gleicher and his colleagues [2003] to semiautomatically build a *fat graph* in which the incoming and outgoing edges of a node represent motion segments starting from and ending at similar poses. Heck and Gleicher [2007] create parametric spaces from similar motion segments and use sampling methods to identify and represent good transitions between these spaces. Our approach is similar in that we also combine motion graphs and interpolation but we do so for off-line, globally optimal search rather than local search.

Unlike all previous motion graph approaches with the exception of Lau and Kuffner [2005], we find a globally optimal or a close-to-optimal solution with an upper bound on the sub-optimality. In Section 7, we show a number of comparisons to demonstrate that globally optimal solutions avoid the inefficient patterns of motion that are often seen with local or sub-optimal search techniques.

## 3 Overview

We represent the motion, $M'(t)$, that we are trying to synthesize as an interpolation of two time-scaled paths through a motion graph:

$$M'(t) = w(t)M_1(t) + (1 - w(t))M_2(t). \quad (1)$$

where $M_1(t)$ and $M_2(t)$ are the paths and $w(t)$ is an interpolation weight. The two paths independently transition between poses in the database (Figure 3). We allow paths to be scaled in time to synchronize the motions for interpolation. The weight, $w(t)$, can also change with time. Equation 1 is very similar to the standard equation for motion interpolation, where $M_1(t)$ and $M_2(t)$ are two short

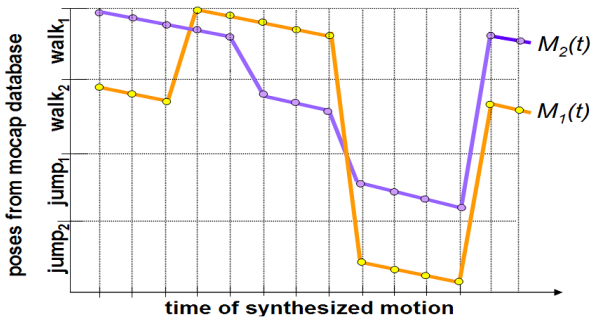



Figure 3: For this example, the database consists of four motions: two walks and two jumps. The resulting motion is an interpolation of two paths through the motion graph, $M_1(t)$ (orange) and $M_2(t)$ (purple). $M_1(t)$ and $M_2(t)$ can transition independently between motions in the database.

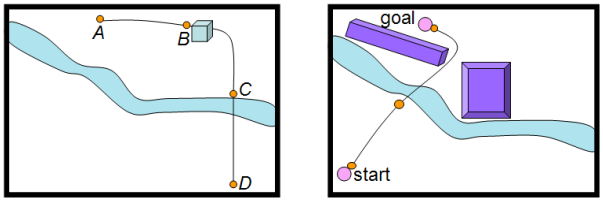



Figure 4: Two example problem specifications. (Left) The user provided the sketch of the path of the character and specified three constraints: start at $A$, pick an object from a table at $B$, and arrive at $D$. An environmental constraint, $C$, for jumping over the river is added automatically by the system. (Right) The user specified only the start and goal positions. The system automatically creates a sketch of the $2D$ path while avoiding obstacles and adds an environmental constraint for jumping over the river.

motion segments of similar structure (two jumps, for example). In our representation $M_1(t)$ and $M_2(t)$ are two long paths through the motion graph which we find using discrete optimization.

We construct a graph that supports interpolation of paths through the original motion graph and use an anytime version of $A^*$ search to find an optimal path that satisfies the constraints specified by the user. We next define the unknowns, constraints, objective function, and search method for our discrete optimization problem.

**Unknowns:** The unknowns of the optimization problem are the variables from Equation 1: $M_1(t)$, $M_2(t)$, and $w(t)$. The weight, $w(t)$ is discretized (eleven values in the range from zero to one).

**Constraints:** Either the user provides a rough sketch of the $2D$ path on the ground plane that the character should follow or the $2D$ path is computed automatically from the start and end points (Figure 4). The root of the character is constrained to stay inside a $2D$ corridor around the path ($0.25-1.0$m wide in the examples reported here). The user also can optionally specify a set of constraints (such as sitting on a chair or picking up an object). These user constraints should coincide with contact changes in the motion and will be met within a small tolerance. If the user sketch passes across obstacles (such as a river) the system also automatically adds environmental constraints. Finally, obstacle avoidance constraints are automatically included.

**Objective function:** The objective function is a weighted average of two terms: the sum of the squared torques computed via inverse dynamics and the sum of the costs of the transitions associated with the traversed edges in the motion graph. The first term is an approximation of the energy needed to perform the motion. This term picks paths through the motion graph whose interpolation will result in efficient motion patterns. The second term is a measure of the smoothness of the motion.

The user-specified constraints are treated as constraints for the optimization problem rather than including them as part of the objective function. This decision makes the objective function independent of the particular constraints chosen by the user at runtime and allows us to compress the motion graph as a preprocessing step (Section 5).

**Search Method:** We use $A^*$ search [Pearl 1984], and in particular its anytime extension $ARA^*$ [Likhachev et al. 2003], to find the paths through the motion graph and interpolation weights so that the interpolated path will satisfy the constraints and provide the optimal solution. The algorithm takes as input a graph where each edge has a strictly positive cost, a start state, $s_{\text{start}}$, and a goal state, $s_{\text{goal}}$. It then searches the graph for a path that minimizes the cumulative cost of the transitions in the path. $A^*$ uses a problem-specific heuristic function to focus its search on the states that are more likely to appear on the optimal path because they have low estimated cost. For each state $s$ in the graph, the heuristic function must return a non-negative value, $h(s)$, that estimates the cost of a path from $s$ to $s_{\text{goal}}$. To guarantee the optimality of the solution and to ensure that each state is expanded only once, the heuristic function must satisfy the triangle inequality: for any pair of states $s, s'$ such that $s'$ is a successor of $s$, $h(s) \leq c(s,s') + h(s')$, where $c(s,s')$ is the cost of a transition between states $s$ and $s'$. For $s = s_{\text{goal}}$, $h(s) = 0$. In most cases, if the heuristic function is admissible (i.e., does not overestimate the minimum distance to the goal), the triangle inequality holds. For a given graph and heuristic function, $A^*$ searches the minimum number of states required to guarantee the optimality of a solution [Russell and Norvig 2003].

The anytime extension of $A^*$, $ARA^*$ search [Likhachev et al. 2003], trades off the quality of the solution for search time by using an inflated heuristic ($h$-values multiplied by $\varepsilon > 1$). The inflated heuristic often results in a speedup of several orders of magnitude. The solution is no longer optimal, but its cost is bounded from above by $\varepsilon$ times the cost of an optimal solution. $ARA^*$ starts by finding a suboptimal solution quickly using a large $\varepsilon$ and then gradually decreases $\varepsilon$ (reusing previous search results) until it runs out of time or finds a provably optimal solution.

## 4 Graph Construction

We assume that we have a database of motions sampled as an ordered sequence of poses. We use a right-handed coordinate system $XYZ$ with the $X$ and $Z$ axes spanning the ground plane and the $Y$ axis pointing up. Each pose is represented by (1) $Q$, the joint angles relative to the inboard link and the orientation of the root around the $X$ and $Z$ axes, (2) $P_y$, the position of the root along the vertical axis, (3) $\Delta P_x$ and $\Delta P_z$, the relative position of the root on the ground plane (computed with respect to the previous pose in this motion sequence) and (4) $\Delta Q_{yaw}$, the relative rotation of the root around the vertical axis (computed similarly).

We first construct a standard motion graph, $MG$, from the database using techniques similar to those in the literature [Li et al. 2002; Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002]. We construct graph $MG$ as a preprocessing step. We then generalize $MG$ to create a motion graph that supports interpolation. We call this graph $IMG$ (interpolated motion graph). Graph $IMG$ can also be constructed as a preprocessing step because it does not require significant space (for our examples $IMG$ requires less than $5MB$). A full search graph, $ISG$, could be constructed by unrolling graph $IMG$ into the environment and augmenting each state with

the global position and orientation of the root. This step is required to search for motions that satisfy user-specified global position constraints and avoid obstacles. Unrolling would cause the size of the graph to grow by the number of possible positions and orientations of the character in the environment because each pose now appears many times in the graph, once for each reachable position and orientation. This graph, therefore, is constructed at runtime as needed.

**Graph MG:** Each state in graph *MG* is defined as $S = (I)$, where $I$ is the index of the pose in the motion capture database. Transitions are constructed by identifying similar poses with the same contact and connecting them with edges as was done by Lee and his colleagues [2002]. However, they only allowed transitions when the character's contact with the environment changed. We instead allow transitions inside of contact phases but prune redundant or non-optimal transitions as a preprocessing step.

**Graph IMG:** Each state in graph *IMG* is defined as $S = (I_1, I_2, w)$, where $I_1$ and $I_2$ are the indices of the two poses to be interpolated and $w$ is the interpolation weight. Constructing graph *IMG* is like taking the "product" of two identical motion graphs. Thus, the maximum number of states in graph *IMG* is $N^2W$, where $N$ is the number of poses in the motion capture database and $W$ is the number of possible weight values. In practice, however, the number of states is much smaller because we interpolate only poses with matching contact states (left foot on the ground, for example). Given state $A$ defined by $(I_1^A, I_2^A, w_1^A)$, we need to compute the set of successor states—the states that can be reached from state $A$ via a transition in the graph *IMG*. State $B$ is a successor of state $A$ if and only if $I_1^B$ is a successor of $I_1^A$, and $I_2^B$ is a successor of $I_2^A$ in the motion graph *MG*.

**Graph ISG:** Graph *ISG* is computed by unrolling graph *IMG* into the environment. Therefore, each state in graph *ISG* is a state in graph *IMG* with additional variables representing the global position and orientation of the character: $S = (I, P_x, P_z, Q_{yaw})$, where $P_x$ and $P_z$ are the global position of the character on the ground plane and $Q_{yaw}$ is the global orientation of the character about the vertical axis. Because we are doing a global search, we need to discretize positions, and orientation into a finite set of values. We discretized positions $P_x$ and $P_z$, into 0.05 by 0.05 meter cells and orientation $Q_{yaw}$, into 1.5 degree intervals. The global variables are computed by interpolating and integrating the relative terms stored in the states of graph *IMG*. In *ISG*, each transition is associated with a cost computed from the objective function.

To ensure that all constraints are satisfied, we add a counter to each state. If a state satisfies the next constraint during a search, its counter is set to that of its predecessor plus one. Because constraints are positioned along the user sketch, the counter also allows states to be pruned from the search if they pass a constraint without satisfying it.

**Reducing Graph ISG:** The complexity of the $A^*$ algorithm is $O(E + SlogS)$, where $S$ is the number of states and $E$ is the number of edges in the graph. If a motion graph, *MG* contains $10,000$ states, the unrolled graph *MG* (without interpolation) will contain $S = 10^{12}$ if we discretize $P_x$ and $P_z$ into 1000 by 1000 values and $Q_{yaw}$ into 100 values. A graph of this size cannot be searched quickly for an optimal solution. As a result, all existing approaches in the literature either find a solution using a global but sub-optimal approach with no guarantee on sub-optimality or search a manually constructed graph with a small number of states. The unrolled, interpolated motion graph, *ISG*, is even more challenging to search because it has a larger number of states ($S = 10^{17}$ assuming we discretize $w$ into 10 values). Constraining the character to stay inside the corridor around the user-specified path would reduce the number of states to $S = 10^{15}$ if 1% of the position values fall within the corridor. This reduction is not enough to make optimal search possible.

To address this problem, we developed two techniques that significantly decrease the number of states that the search will need to visit. The first technique compresses the motion graph into a practically equivalent but much smaller graph. The second technique computes an informative heuristic function that guides the search toward states that are more likely to appear in an optimal solution. In Section 7, we show that the combination of these techniques makes it possible to find an optimal or a close-to-optimal solution for a database of a reasonable size with a few minutes of computation. The next two sections give the details of both techniques.

# 5 Graph compression

We compress the motion graph in two steps. First, we cull states and transitions that are sub-optimal. These states will not appear in any optimal solution because the graph contains a lower cost alternative. Second, we cull states and transitions that are redundant because they are similar to other states and transitions in the motion graph. These steps result in a compressed version of graph *MG* and the graph *IMG* is derived from that graph as described in Section 4.

**Culling sub-optimal states and transitions:** To cull transitions, we first identify a specific class of states: those in which a contact change occurs (from double support to right leg contact, for example, or from no object in hand to object in hand). More formally, state $S$ in motion $M$ is defined as a *contact change state* if the state that directly precedes state $S$ in motion $M$ has a different set of contacts with the environment. To determine the contact change states, we separate motions into phases based on contact with the environment. We use the technique of Lee and his colleagues [2002] to identify the contacts and then verify them by a visual inspection (only a very small percentage of the contacts need to be adjusted by hand for locomotion and other simple behaviors). Contact information could also be computed using one of the other published techniques [Ikemoto et al. 2006; Callennec and Boulic 2006]. These algorithms identify contacts as not moving with respect to the ground plane or contact object and we rely on that property in our graph compression algorithms.

We can then compute paths that connect pairs of contact change states without passing through another contact change state. All states in each such path will have the same contacts except for the terminal state where the contact changes. We call these paths *single contact* paths.

We can remove a large number of single contact paths from the graph *MG*. The key insight behind our algorithm is that although there are likely to be many single contact paths that connect two contact change states (thousands in our experiment), they all end with the character in exactly the same pose and with the same root position and orientation (Figure 5). Only one of these paths is optimal with respect to our optimization function. Therefore we can cull all other paths before unrolling the graph into the environment without reducing the functionality of the graph. Figure 6 illustrates this process.

This culling step does not affect the functionality of the graph unless the constraints provided by the user require controlling the details of the motion during a period of time when the contacts are not changing. For example, the user could no longer ask for waving while standing in place. Because animated characters tend to act on their environment, user constraints often create contact changes we have not found this restriction to be a serious problem. We can revert to searching an uncompressed motion graph if a constraint falls in the middle of the contact phase.

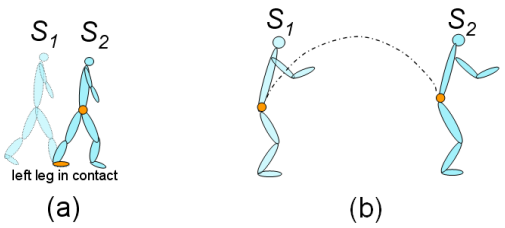



Figure 5: (a) For direct paths between a pair of contact change states $S_1$ and $S_2$, the global position and orientation of the root of the character at state $S_2$ is uniquely determined by the contact position and orientation at state $S_1$ and the values of the joint angles at state $S_2$. (b) The position of the center of mass at landing (state $S_2$) is uniquely defined by the intersection of the flight trajectory and the center of mass of the character at state $S_2$. The trajectory of the center of mass for the root of the character is defined by the lift-off velocity from state $S_1$.

The optimal path might also violate environmental constraints if the swept volume for the character from one contact change state to the next intersects an obstacle. If the original graph contains a different path that would not have violated the constraints, then the culled graph will have lost functionality. This situation is uncommon because neither endpoint intersects the obstacle (or the search would not have explored the state) and only limited movement is possible with one contact change.

The optimization function we use allows us to compute optimal paths as a precomputation step because it is independent of the particular constraints the user specifies. Many common optimization functions are similarly independent of the problem specification: minimizing energy, minimizing sum of squared accelerations, maximizing smoothness, minimizing the distance traversed, minimizing the total time of the motion, and satisfying specified annotations (for example behaviors or styles as in [Arikan et al. 2003]). We can also support objective functions that depend on the user specification at contact change states. These functions can often be used to approximate other functions. For example, instead of minimizing the distance between every frame of the motion and the user-specified sketch, we can minimize the distance between the contact change states and the sketch.

Culling transitions in this way is different from retaining only the transitions between contact change states, as others have done [Lee et al. 2002]. With that approach no path would be found between states $A$ and $D$ in Figure 6(a) even though one exists in the original motion graph. The preprocessing presented here retains many more unique transitions, a property that is important for finding transitions between different behaviors such as a walk and a jump.

**Culling redundant states and transitions:** After we cull the sub-optimal states and transitions, we cull redundant ones. Motion graphs often include redundant data because of the need to capture natural transitions between behaviors. For example, motion capture data for walking and jumping transitions includes many similar walking steps as part of the lead-in to the jump. As a result, many states in the motion graph will be similar. Removing this redundancy significantly reduces the size of the graph. We begin with the output of the previous compression step, a graph that contains only contact change states and transitions that are optimal sequences of poses between contact change states.

We merge all similar states in the order of their similarity. For example, state $A$ in Figure 7 has three successors, $S_1$, $S_2$ and $S_3$,

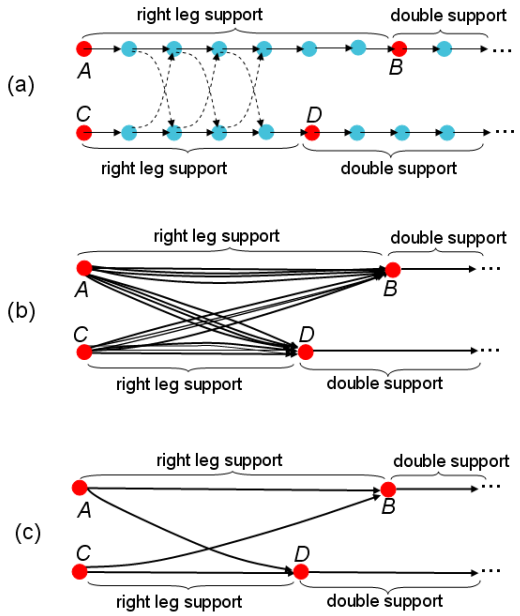



Figure 6: (a) States $A$, $B$, $C$ and $D$ (shown in red) are contact change states. If the character enters state $A$ (and initiates a right leg support phase), it can exit only through state $B$ or $D$. (b) A representation with only the contact change states shows that there are many paths between each state. (c) The graph after transitions are culled to include only optimal paths between contact states.

that are very similar to each other. Because the states have a common predecessor and similar poses, the transitions leading to these states will end at approximately the same position in the environment when the graph is unrolled. Therefore, these transitions are redundant.

When two or more states are merged to form a new state, the successors of that state are the union of the successors of the merged states. Similarly, the predecessors are the union of the predecessors of the merged states. Therefore after merging, we will have multiple transitions connecting a single pair of states. If we use a low threshold for merging, then all these transitions will end in approximately the same place in the environment when the graph is unrolled and we can just keep one, lowest cost transition without affecting optimality (Figure 7). However, we can get better compression of the graph if we choose a higher threshold for merging states and allow larger changes in pose between the merged states. The consequence of this more aggressive merging threshold is that the transitions no longer necessarily place the character in the same position in the environment when the graph is unrolled. We therefore delete only transitions which result in very similar positions to retain the flexibility of the original graph. Because each transition in the compressed graph is a sequence of poses representing one contact phase of the motion, we can process each transition to remove foot-sliding as a preprocessing step.

**Constructing $IMG$ from the compressed $MG$:** After graph $MG$ is compressed, graph $IMG$ is constructed from it as described in Section 4. In the compressed graph $MG$, however, each transition is a sequence of poses in between two contact change states. Consequently, a transition from state $A = (I_1^A, I_2^A, w_1^A)$ to state $B = (I_1^B, I_2^B, w_1^B)$ in graph $IMG$ is now a sequence of poses where each pose is an interpolation of corresponding poses in the transitions from $I_1^A$ to $I_1^B$ and from $I_2^A$ to $I_2^B$ in the compressed $MG$ (Figure 8).

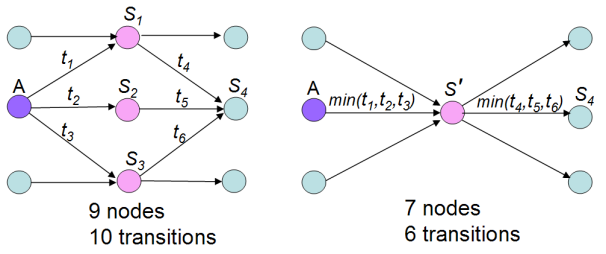



Figure 7: (Left) States $S_1$, $S_2$ and $S_3$ are similar to each other. As a result, transitions $t_1$, $t_2$ and $t_3$ all end with the character at approximately the same position. (Right) We merge states $S_1$, $S_2$ and $S_3$ into one state $S'$ and keep only the lowest cost transition.

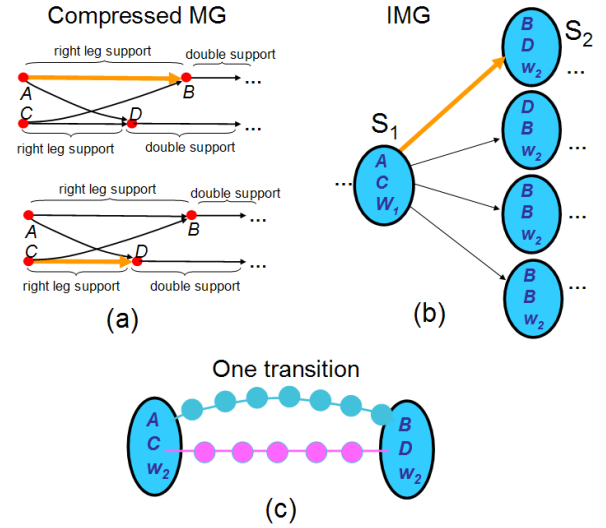



Figure 8: (a) Two identical versions of the compressed graph *MG*. (b) *IMG*. The transition between states $S_1$ and $S_2$ in graph *IMG* is an interpolation of a path from *A* to *B* and a path from *B* to *D* in the compressed graph *MG*. These two paths are shown by thick arrows in (a). (c) Shows these two paths in more detail with circles representing the frames. Because the paths can be of different length, we uniformly scale them in time.

The interpolation weight $w$ is constant throughout the transition. We use the interpolation scheme described in Safonova and Hodgins [2005]. In particular, (1) during flight we interpolate the center of mass position (instead of the root position) and all the joint angles; (2) during ground contact we interpolate the positions of the feet, the center of mass position, and all nonredundant degrees of freedom to prevent the feet from sliding on the ground. When the durations of the transitions from $I_1^A$ to $I_1^B$ and from $I_2^A$ to $I_2^B$ differ, we assume a uniform time scaling with the time of the interpolated segment computed according to the following equation:

$$T = \sqrt{T_1^2 w + T_2^2 (1-w)} \qquad (2)$$

where $T_1$ and $T_2$ are the time durations of the first and second segments being interpolated.

Safonova and Hodgins [2005] showed that this interpolation scheme ensures that the majority of the interpolated transitions in graph *IMG* are close to physically correct. We can also check these interpolated transitions for physical correctness using inverse dynamics when graph *IMG* is constructed.

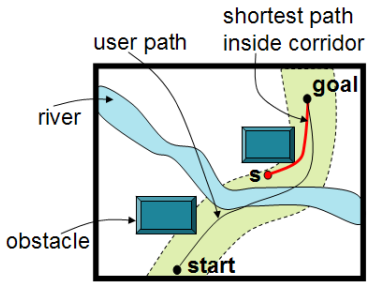



Figure 9: $H_{pos}(S,G)$ is the shortest path from the position of the character at state $S$ to the goal. The shortest path is constrained to stay inside the corridor.

## 6 Informative heuristic function

We use an anytime version of the $A^*$ search algorithm to find an optimal path in the unrolled graph, *ISG*. The number of states that the $A^*$ search explores depends on the quality of the heuristic function—the lower bounds on cost-to-goal values. Informative lower bounds can significantly reduce the amount of the search space that is explored. In this section, we present a method for computing such bounds. In Section 7, we show that this heuristic function usually speeds up the search by several orders of magnitude and is often the determining factor in whether a solution can be found.

The heuristic function must estimate the cost of getting to the goal while satisfying user and environmental constraints for each state $S$ in the graph *ISG*. We compute two heuristic functions: $H_{pos}$ and $H_{img}$. The first heuristic function, $H_{pos}$, ignores the dynamics of the motion of the character and estimates the cost of getting to the goal based only on the current position of the character, sketch of the user path, and obstacles in the environment. The second heuristic function, $H_{img}$, takes into account the capabilities of the character that are encoded in the motion graph but ignores its position in the environment. The combination of the two heuristics creates an informative measure of the cost of solving the problem specification. We now describe how to compute $H_{pos}$ and $H_{img}$ and how to combine them.

**Heuristic function based on the character location ($H_{pos}$):** $H_{pos}(S,G)$ is the shortest path on the ground plane from the position of state $S$ to the position of the goal state $G$. The path must avoid obstacles and remain inside the corridor around the user-specified path (Figure 9). Because the computation of the heuristic $H_{pos}(S,G)$ depends on the user's sketch, it must be computed at runtime.

To compute $H_{pos}(S,G)$, we discretize the environment into 0.2 by 0.2 meter cells and compute the shortest path from the center of each cell to the goal. A single Dijkstra's search on a $2D$ grid can be used to compute all such paths with only a few milliseconds of computation. Because our cost function minimizes a weighted average of torque and smoothness terms, we need to multiply the shortest distance (in meters) by an estimate of the minimum objective function value required to traverse one meter. We compute this minimum value from the motion graph data using inverse dynamics for the torques and the transition cost for smoothness. We use a coarse discretization to compute the heuristic function, but a much finer discretization when computing the unrolled graph *ISG* to avoid discontinuities in the final motion (0.2m vs. 0.05m).

**Heuristic function based on motion graph state ($H_{img}$):** $H_{pos}(S,G)$ provides a reasonable estimate of the cost to the goal for motions that simply require the character to travel from one location

in the environment to another. But if user or environmental constraints are present then $H_{pos}(S,G)$ will underestimate the cost to the goal because satisfying constraints usually requires much more effort than the minimum torque estimate assumed by $H_{pos}$. The motion graph also restricts what the character can do from a particular state, perhaps making a state that satisfies the constraint hard to reach. For example, if it is difficult to reach a jumping motion from state $S$, then the cost-to-goal at state $S$ should be high when there is a jumping constraint. $H_{pos}$ will underestimate this cost and, consequently, the $A^*$ search will needlessly explore this part of the space. The second heuristic function, $H_{img}$, addresses this problem by taking into account the capabilities of the character that are encoded in the motion graph. It estimates the cost of satisfying each type of constraint for each state in the interpolated motion graph.

We support five types of constraints: picking, jumping, stepping onto an obstacle (a beam for example), ducking, and sitting. For each type of constraint, $H_{img}(S,C)$ is computed as the minimum cost of getting to any pose in the interpolated motion graph that satisfies constraint $C$ from state $S$. To avoid double counting the "distance traveled" cost already estimated by $H_{pos}$, we compute $H_{img}$ after first subtracting from each transition in graph IMG the minimum cost required to traverse the planar distance covered by that transition.

The computation of the $H_{img}$ heuristic does not depend on the particular constraint specified by the user and therefore can be precomputed. The computation is automatic if the contact information for all motions in the motion capture database is available. We use the constraint of picking up an object to explain the computation of $H_{img}$. The same method is used for all constraints supported by our system and should generalize to other types of constraints such as kicking, stepping over obstacles, or standing on one leg.

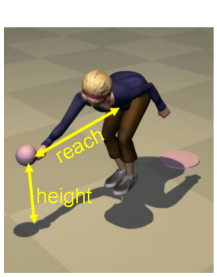
Each "picking" pose can be defined by two parameters: *height* and *reach* (Figure 10(a)). *Height* is the height of the object with respect to the ground. *Reach* is how far the character must reach out to pick up the object (distance between the root and the hand projected onto the ground). Based on the contact information, we automatically identify each state in the graph *IMG* that represents picking up an object $p = (I_1, I_2, w)$, where both poses, $I_1$ and $I_2$, are states where an object was picked up. At this state, the character assumes a picking pose with *height* and *reach* values based on the interpolation of poses $I_1$ and $I_2$ with weight $w$. For each state in graph *IMG*, we then compute $H_{img}(S,C)$ as a table (Figure 10(b)) where each cell represents a range of *height* and *reach* values, and the value is the minimal cost of getting from the given state to a state with *height* and *reach* values in this range. For the database of 6-7 minutes of motion, the precomputation of the $H_{img}$ heuristic for all constraints took less than an hour.

**Combining the two heuristics:** We combine $H_{pos}(S,G)$ and $H_{img}(S,C)$ into a single heuristic function by summing them together. If at state $S$ there are $n$ constraints left to be satisfied, we fetch the $H_{img}$ term for each of these constraints, and then add all of them to the $H_{pos}(S,G)$ term to obtain a heuristic value for state $S$:

$$H(S) = H_{pos}(S,G) + \sum_{i=1...n} H_{img}(S,C_i) \qquad (3)$$

## 7 Experimental results

To illustrate the effectiveness of our approach, we generated a variety of examples. For each experiment, the user specified a 2D path in the environment that the character should follow and the width of the corridor around that path. In some experiments, the user also specified constraints such as picking up an object or sitting on a chair. Based on the sketch and the current environment, the system automatically computed environmental constraints such as stepping onto an obstacle, ducking, and jumping. The motions for all the experiments are shown in the accompanying video.




(a)

state $S_i$

| height \ length | 0.2-0.4 meters | 0.4-0.6 meters | 0.8-1.0 meters | 1.0-1.2 meters |
|---|---|---|---|---|
| 0.2-0.4 meters | cost | cost | cost | cost |
| 0.4-0.6 meters | cost | cost | cost | cost |
| 0.8-1.0 meters | cost | cost | cost | cost |
| 1.0-1.2 meters | cost | cost | cost | cost |

(b)

Figure 10: (a) We identify states where objects are picked up in the motion graph. Each such pose is parameterized by two parameters: *height* and *reach*. (b) For each state in the motion graph we precompute a table with the minimal cost of getting to a "picking" state with the specified height and reach parameters.

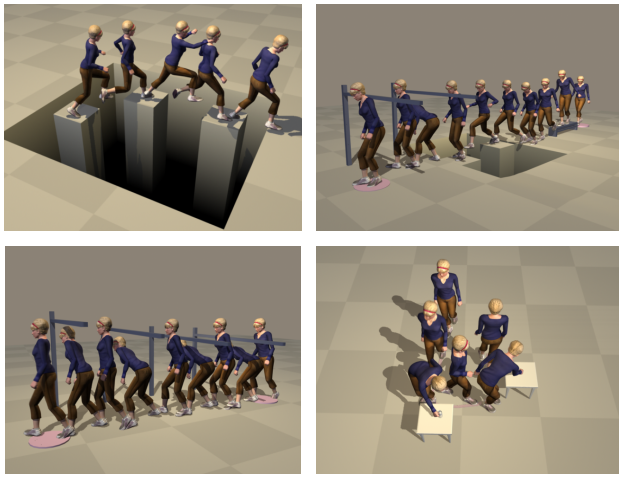
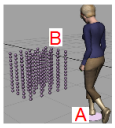
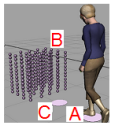


Figure 11: Synthesized motions

Figure 2 shows the motion of a character traversing an obstacle course. The character walks over the beam, jumps over holes, ducks under a bar, and finally sits on a chair. This example illustrates that our algorithm can synthesize motions that are 15 seconds long and consist of several different behaviors. Besides the obstacle course examples, we have also synthesized many other examples, including walking along paths of varying curvature, picking and placing an object in various locations, jumping over stones with variable spacing, jumping with different amounts of rotation and distance, and forward walks of different step lengths. Figure 11 shows images for some of the results. For shorter, single behavior examples, such as jumps and short walks, only a few milliseconds to a few seconds were required to compute an optimal solution. For longer, multi-behavior examples, a few minutes were required to compute a close-to-optimal solution. In general, the time depends on the size of the database, the length of the generated motion, and the complexity of the constraints. We next describe a number of experiments designed to assess the performance of our algorithm.




| | Error Tolerance | | |
|---|---|---|---|
| | 2.5 cm | 5 cm | 10 cm |
| no interpolation | 31% | 70% | 99% |
| interpolation | 99% | 100% | 100% |




| | Error Tolerance | | |
|---|---|---|---|
| | 2.5 cm | 5 cm | 10 cm |
| no interpolation | 16% | 45% | 87% |
| interpolation | 96% | 99.5% | 100% |


| | Error Tolerance | | |
|---|---|---|---|
| | 2.5 cm | 5 cm | 10 cm |
| no interpolation | 40% | 75% | 100% |
| interpolation | 100% | 100% | 100% |

Figure 12: The success rate for three test problems for three different error tolerances and interpolation/no interpolation. (Top) The character needs to start at *A* and pick up an object at *B*. (Middle) The character needs to start at *A* and pick up an object at *B* but now we also constrain the root position of the character to *C* while picking up the object. (Bottom) The character needs to start at *A* and walk to *B* with one stride. We sampled the locations of constraint *B* (179 samples for first two problems and 10 samples for the third problem). An experiment is counted as a success if the search finds a solution within two minutes and within a specified error tolerance.

### 7.1 The benefit of interpolation

The first experiment demonstrates that interpolation in conjunction with a motion graph allows us to satisfy user-specified constraints with a small error tolerance. We tested on two specifications of picking up an object and one of walking. With interpolation, the system consistently finds solutions for small error tolerances but without interpolation, only much higher error tolerances produced consistent results (Figure 12). We also computed the average cost of the successful solutions for the second problem. The average solution cost is about 1.35 times greater without interpolation, with higher costs for small tolerances. Without interpolation, the search has much less freedom to satisfy the constraints and the solution is more likely to contain dithering and inefficient motion patterns that result in a higher solution cost (the video shows a few examples).

Simultaneously selecting two paths for interpolation generates a better result in many situations than the greedy approach of first finding an approximate solution without interpolation and then warping this solution to satisfy the user-specified constraints. Figure 13 shows a situation where a solution that satisfies all user-specified constraints can be found without interpolation. Without interpolation, the character jumps between the columns but with interpolation, the character steps between the first two columns and jumps from the second to the third. Because the optimal strategies differ, we cannot warp one solution into the other.

Figure 14 shows a situation where, without interpolation, there is no solution that satisfies all the user-specified constraints. In this example, the character needs to follow a curve within the user-specified tolerance (represented as a corridor around the curve). With interpolation, the character accurately tracks the curve and takes six steps to reach the goal. Without interpolation, we have to increase the corridor width to find a solution and the character takes only five steps. Because the two solutions use a different number of steps, it would not be possible to warp one into the other. For some constrained situations, it may not even be possible to increase the tolerance without violating the problem specification (missing a stepping stone entirely, for example).

Figure 13: (Left) The optimal solution with interpolation. The character walks across the first column and jumps across the second column. (Right) The optimal solution without interpolation. The character jumps between each pair of columns.

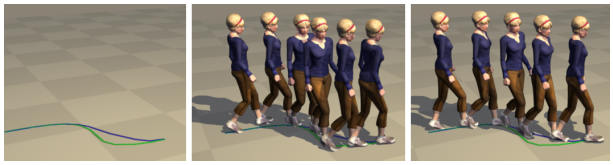


Figure 14: (Left) The user wants the character to walk along the green curve. The blue curve shows the trajectory of the root for the best solution without interpolation. (Middle) The solution with interpolation. The character can track the curve, deviating only slightly from the prescribed path. (Right) The solution without interpolation. The corridor width had to be increased to find a solution and the character takes five steps, whereas with interpolation, she takes six steps.

### 7.2 The benefit of optimality

In the accompanying movie, we show several examples that illustrate that globally near-optimal solutions avoid the dithering and inefficient patterns of motion that sub-optimal solutions often have. Figure 15 shows the results for two motions: a walk to a place where the character needs to pick up an object and a walk from the start to the goal. As the optimality of the solution increases, the character finds more efficient motion patterns.

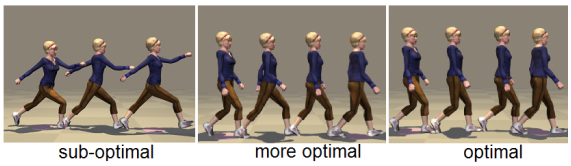
### 7.3 The benefit of motion graph compression

In this experiment, we evaluate the effect of motion graph compression. Table 1 shows these statistics for three different databases: (1) walking, jumping, ducking, sitting and walking along a beam; (2) walking and picking up an object; (3) just walking motions. For each database, we computed the number of states and transitions in the motion graph before compression, after the first compression step (removing sub-optimal data), and after the second compression step (removing redundant data). The table also gives the time

| | Original motion graph | After removing sub-optimal data | After removing redundant data | Compression time |
|---|---|---|---|---|
| DB 1 | states=6,000 trans=90,000 | states=350, trans=12,500 | states=130 trans=700 | 30 min |
| DB 2 | states=12,000 trans=250,000 | states=700 trans=60,000 | states=300 trans=5,000 | 60 min |
| DB 3 | states=2,000 trans=25,000 | states=173 trans=3,700 | states=50 trans=300 | 2 min |

Table 1: Compression for three motion graphs. The first graph is computed from motions of walking, jumping, ducking, sitting and walking along the beam. The second graph is computed from motions of walking and picking up an object and the third one is computed from just walking motions.


| Search Time (secs) | Solution Cost | Optimality Bound ($\varepsilon$) |
|---|---|---|
| 0.67 | 2,700,000 | 10.0 |
| 3.42 | 1,800,000 | 2.0 |
| 50.00 | 1,600,000 | 1.0 |




| Search Time (secs) | Solution Cost | Optimality Bound ($\varepsilon$) |
|---|---|---|
| 0.70 | 1,200,000 | 10.0 |
| 9.10 | 650,000 | 2.0 |
| 20.10 | 550,000 | 1.0 |

Figure 15: (Left): the character walks to a small object; the first solution requires an unnatural posture; the second solution is better but the character reaches in from the side to contact the object; the optimal solution looks natural. (Right): walk from start to goal; the first solution is visually very suboptimal—the character takes two awkwardly large steps to reach the goal position; the second solution is better—the character takes smaller steps but the walk is a bit unnatural because the steps are of different length; the final solution is optimal and looks natural. The tables show the computation time, solution cost and the sub-optimality bounds for each of the results.

required to compress the graph (a precomputation step performed only once for each database). Compression techniques reduce the size of the graph by a factor of 20 to 50.

### 7.4 The benefit of the heuristic function

We also evaluated the effectiveness of our heuristic function. The results are shown in Table 2. We compare four heuristics: (1) the Euclidean distance to the goal; (2) the $H_{pos}$ component of our objective function; (3) the $H_{img}$ component of our objective function; (4) the combined heuristic function with both $H_{pos}$ and $H_{img}$ components. The results demonstrate that our heuristic function is essential for making the search efficient and often makes the difference between finding a good solution and not finding one at all. The table also shows that both components of the heuristic function are important, neither component alone is effective.

## 8 Discussion

In this paper, we described a discrete reduced-space representation of human motion. This representation can be viewed as a combination of motion graph and interpolation techniques as it generates motion that is an interpolation of two time-scaled paths through a motion graph. Finding a solution in this smaller search space is much easier than finding a solution in the 50 degree of freedom search space traditionally used to represent human motion. In addition, the synthesized motion is likely to contain the natural coordination patterns of the original data. We have shown that optimization in this discrete space allows for the synthesis of long, less dynamic motions that are composed of different behaviors. It takes a few minutes to compute a 15 second long motion that is close to optimal from a database of 6-7 minutes of motion.

Finding a close-to-optimal solution is often important for obtaining high quality motion. This property holds not only for interpolated motion graphs but also for regular motion graphs. Our compression techniques and informative heuristic should apply equally well to regular motion graphs.

We allow interpolation of arbitrary poses as long as they have the same contacts. This restriction, together with the minimization of squared torques was enough to select motion segments whose interpolation resulted in natural looking motion. Perhaps this behavior is related to the observation that continuous optimization can often find a natural looking motion by minimizing the sum of squared torques. For example, two walk segments with both arms at the character's side are more likely to be selected for interpolation than a segment with one arm waving because the latter requires more energy. For bigger databases, we may need to explicitly restrict the segments that can be interpolated. For example, we could interpolate only poses with the same behavior label (using a labeling algorithm from the literature).

Because our method computes a compressed motion graph that contains only optimal paths, variations in the original data are "suboptimal" and will be culled. We would like to experiment with keeping several maximally different paths rather than just one. In our experience, most of what is culled are redundant trajectories that are visually indistinguishable but additional experiments would be required to decide whether important variability is lost.

The quality of the results largely depend on the quality of the motion database used to construct the motion graph. For example, if the database contains only a motion of sitting on a tall chair then we cannot synthesize a motion for sitting on a medium or a low height chair because there are no two motions whose interpolation would provide the desired motion.

We also found that the motion graph must have "good" connectivity. For example, if it is impossible to reach a state where an object can be picked up from the other states in the motion graph, then we cannot synthesize motions that satisfy such a constraint. Our experiments show that to obtain good results many states must be able to quickly connect to the constraint states and vice versa. An automatic technique for evaluating the connectivity of a motion graph would be very helpful. Reitsma and Pollard [2004] evaluated the quality of a motion graph for navigational tasks. Extending this evaluation to our domain would be very useful.

Better automatic methods for constructing motion graphs with "good" connectivity would also be very helpful [Ikemoto et al. 2007]. We found that a single threshold for picking good transitions often does not work well. A low threshold results in most transitions occurring within a single behavior (walks for example) and very few transitions between motions of different behaviors (walks and jumps, for example). A high threshold, on the other hand, results in many low quality transitions within a single behavior even though these transitions are not needed.

Our experimental results show that our approach works well for a database with 6-7 minutes of motion. This result is comparable to the databases used in motion graph papers: [Lee et al. 2002] used 5-1 0minutes, [Arikan et al. 2003] used 7 minutes, [Kovar et al. 2002] used 3-4 minutes. Scaling our approach to a larger database, however, will require additional work. We plan to experiment with automatic clustering of motions into behaviors and interpolating mo-

| $\varepsilon$ | Euclidean distance | | | $H_{2D}$ | | | $H_{mg}$ | | | $H_{combined}$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | time | exp | solved | time | exp | solved | time | exp | solved | time | exp | solved |
| 10.0 | 8.0 | 185,813 | 100% | 8.1 | 160,718 | 100% | 11.6 | 72,004 | 100% | 0.8 | 9,332 | 100% |
| 3.0 | 17.1 | 481,321 | 100% | 16.8 | 406,149 | 100% | 15.1 | 103,000 | 100% | 1.6 | 16,068 | 100% |
| 1.0 | 100.2 | 1,832,347 | 20% | 97.8 | 1,748,620 | 20% | 48.1 | 270,812 | 80% | 49.5 | 275,712 | 80% |

Table 2: Evaluation of the heuristic function for the problem of picking up an object. We sampled the location of the object into 179 samples. Each column shows the average search time in seconds, the average number of states expanded during the search and the percent of the experiments that succeeded (found solution within 10 minutes and did not run out of memory). The statistics are reported for the Euclidean distance to the goal, the $H_{2D}$ component alone, the $H_{mg}$ component alone, and the combined heuristic function. The first row shows results for a solution whose cost is at most 10 times the optimal one. The sub-optimality bound for the second row is 3 and the solution in the last row is optimal.

tions only within the same class to further reduce the size of the problem and allow the use of larger databases.

In all of our experiments, the interpolation of two paths was sufficient to find a solution that met the user's constraints and produced natural-looking motion. Some problems, however, may require the interpolation of more than two paths in order to satisfy the constraints. We would like to scale our approach to the interpolation of three paths. If this proves to be infeasible, we can also try an iterative approach: first, compute the best possible solution for the interpolation of two paths; then compute a second solution for the interpolation of two paths that when interpolated with the solution we already have, produces a better result; continue in this manner until no further improvement is possible. This iterative, greedy, approach did not work when the first solution did not include interpolation, but if the interpolation of two paths is sufficient to identify the right sequence of behaviors (strategy), then small refinements via the greedy approach should work well.

Our approach can currently find a close-to-optimal solution for motions that are approximately 15 seconds long. As the length of the motion increases, the complexity of the search increases. We can decrease the complexity by limiting the number of times interpolation is used in the final solution. Interpolation is most often required near user-specified constraints (such as the position for picking up the object or a sharp turn of the user path) and is generally unnecessary when the character is moving in free space. We can limit interpolation by adding another variable to each state in the graph to count the number of motion segments that have been interpolated so far. We can then limit this variable during the search and to control the amount of interpolation. We have run preliminary experiments with this approach and have found that it can greatly reduce the complexity of the search.

## 9 Acknowledgments

The authors would like to thank Moshe Mahler for his help in modeling and rendering and Justin Macey for his assistance in collecting and cleaning the motion capture data. The authors would like to thank Autodesk for their donation of Maya software. This material is based upon work supported by the National Science Foundation under Grant No. IIS-0326322.

## References

ABE, Y., LIU, C. K., AND POPOVIĆ, Z. 2004. Momentum-based parameterization of dynamic character motion. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 173–182.

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

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

CALLENNEC, B. L., AND BOULIC, R. 2006. Robust kinematic constraint detection for motion data. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 281–290.

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

FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Trans. on Graphics 22*, 3, 417–426.

GLEICHER, M., SHIN, H., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: Assembling run-time animation. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 181–188.

GUO, S., AND ROBERGE, J. 1996. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*, 95–107.

HECK, R., AND GLEICHER, M. 2007. Parametric motion graphs. In *ACM Symposium on Interactive 3D Graphics*, 129–136.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2006. Knowing when to put your foot down. In *ACM Symposium on Interactive 3D Graphics*, 49–53.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2007. Quick transitions with cached multi-way blends. In *ACM Symposium on Interactive 3D Graphics*, 145–151.

KOVAR, L., AND GLEICHER, M. 2003. Flexible automatic motion blending with registration curves. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 214–224.

KOVAR, L., AND GLEICHER, M. 2004. Automated extraction and parameterization of motions in large data sets. *ACM Trans. on Graphics 23*, 3, 559–568.

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

KWON, T., AND SHIN, S. Y. 2005. Motion modeling for on-line locomotion synthesis. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 29–38.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 271–280.

LAU, M., AND KUFFNER, J. 2006. Precomputed search trees: Planning for interactive goal-driven animation. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 299–308.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 79–87.

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 Trans. on Graphics 21*, 3, 491–500.

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

LIKHACHEV, M., GORDON, G., AND THRUN, S. 2003. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*, Cambridge, MA: MIT Press.

PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 105–112.

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

PEARL, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics 1*, 1, 5–15.

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: texturing and synthesis. *ACM Trans. on Graphics 22*, 2, 501–508.

REITSMA, P. S. A., AND POLLARD, N. S. 2004. Evaluating motion graphs for character navigation. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 89–98.

ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications 18*, 5, 32–40.

RUSSELL, S., AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall.

SAFONOVA, A., AND HODGINS, J. K. 2005. Analyzing the physical correctness of interpolated human motion. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 171–180.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. on Graphics 23*, 3, 514–521.

SHIN, H. J., AND OH, H. S. 2006. Fat graphs: Constructing an interactive character with continuous controls. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 291–298.

SRINIVASAN, M., METOYER, R. A., AND MORTENSEN, E. N. 2005. Controllable real-time locomotion using mobility maps. In *Proc. of Graphics Interface*, 51–59.

SULEJMANPAŠIĆ, A., AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Trans. on Graphics 24*, 1, 165–179.

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. In *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, 291–300.

WILEY, D. J., AND HAHN, J. K. 1997. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics Applications 17*, 6, 39–45.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. *Computer Graphics (Proceedings of SIGGRAPH 88) 22*, 4, 159–168.