Construction and optimal search of interpolated motion graphs

Alla Safonova **Jessica K. Hodgins**

January 2007
CMU-CS-07-106

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Many compelling applications would immediately become feasible if novice users had the ability to synthesize high quality human motion based only on a simple sketch or 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^* to find a globally optimal solution in this graph that satisfies the user's specification. This solution is not subject to local minima and avoids the inefficient motions that are sometimes seen with locally optimal search algorithms and traditional motion graphs. 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.

This research was sponsored by the National Science Foundation under grant nos. CNS-0196217, IIS-0205224, and IIS-0326322 and supported by generous software donations from Alias/Wavefront and Autodesk.

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

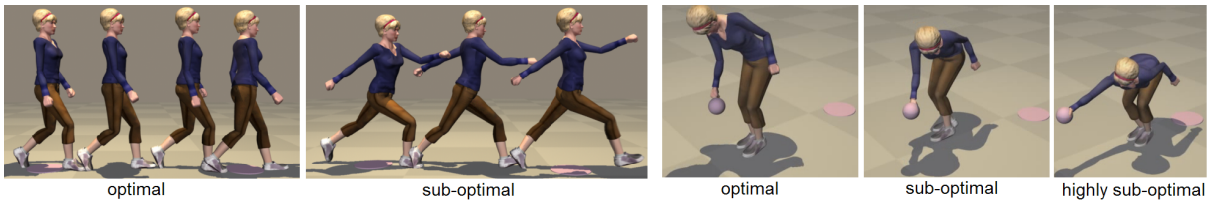


Figure 1: Optimal and greedy solutions for walking a given distance and for picking up an object.

1 Introduction

The ability to construct animations of human characters easily and without significant training would allow the creation of many novel and compelling applications. Children could animate their stories, witnesses could visually describe an accident, and football fans could re-tell their favorite plays. With these applications and others like them in mind, we have focused on techniques that require users to provide only a small amount of information about the desired motion. Given a sketch of the desired motion or a few constraints and an appropriate motion capture database, our discrete optimization approach can quickly create a natural looking motion that matches the user's specification (Figure 2).

The key insight behind our approach is that we represent the motion as an interpolation of two time-scaled paths through a modified motion graph. This representation creates a compact search space that favors natural poses, velocities, and transitions while also allowing the adaptation of existing motions through interpolation. In addition to the advantages obtained by combining motion graphs and interpolation, we are also able to obtain an optimal or near-optimal solution by using an anytime version of A^* [19].

A naive construction of a motion graph that supports interpolation would be much larger than a traditional motion graph. In order to obtain an optimal solution, we automatically cull redundant or non-optimal states and transitions and implement an informative heuristic function that guides the search toward states that are more likely to appear in an 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 [16]. Figure 1 shows two examples of the motion generated by our system.

We demonstrate the power of this approach with a number of examples. The examples are lengthy (up to 15 sec) 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. The value of adding interpolation is demonstrated through a set of examples in which the constraints can only be achieved through interpolation because the original database does not include a motion that matches the constraints. The value of an optimal solution is shown through examples such as those in Figure 1 where the greedy solution is qualitatively different from the optimal solution. The greedy solution utilizes a less efficient behavior or takes an unnaturally small number of steps, for example.

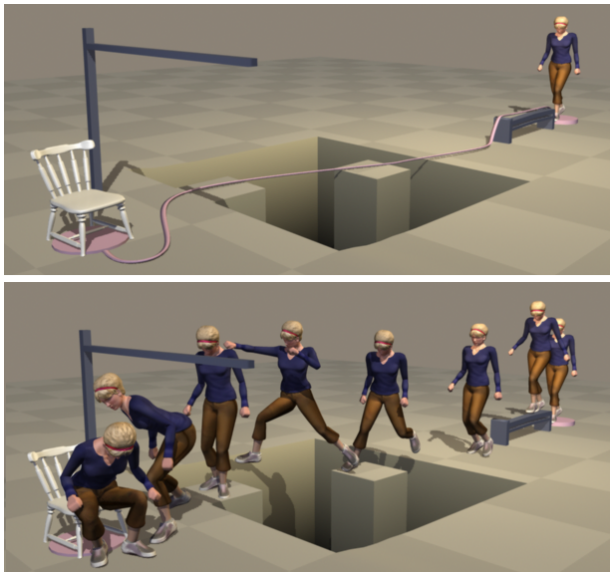


Figure 2: Top: The desired path and constraints. Bottom: Synthesized motion.

In the next section we review the related work. We then define the optimization problem and explain how the graph is constructed and searched efficiently. We conclude with experimental results in Section 6 and a discussion of the advantages and limitations of our approach in Section 7.

2 Background

Continuous optimization, introduced to the graphics community by Witkin and Kass [35], 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 [6, 29, 32]. In contrast to continuous optimization, the discrete optimization approach explored in this paper can handle longer motions consisting of multiple behaviors where no initial guess is available. However, it is restricted to the resequencing and interpolation of pre-recorded motions and cannot generate entirely "novel" motions as continuous optimization can.

Interpolation is a simple and yet powerful technique for generating motions that are variations of motions in a database [23, 34, 8]. Rose and his colleagues [26] implemented a system that allowed interactive control of a character driven by interpolated motion. Kovar and Gleicher [11, 10] created a technique for identifying motions that could be interpolated and automatically registering those motions. Abe and his colleagues [1] 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 [28] analyzed interpolated motions for physical correctness and showed that interpolation produces motions that are close to physically correct in many cases. Like

these techniques, our discrete optimization approach relies on interpolation to synthesize variations of existing motions, but it does not require that the example motions be aligned in advance. Instead the subsequences that are interpolated are selected automatically during the optimization process allowing us to generate long multi-behavior motions. In addition, the optimization process automatically selects what behaviors to use and when to transition from one behavior to another.

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) [12, 16] and *off-line* approaches where the full motion specification is known in advance [2]. On-line approaches can only perform a local search as new input is continuously arriving while off-line approaches can perform a more sophisticated search to find a high quality solution that minimizes an objective function such as energy. Our work falls into the category of off-line techniques. In particular, we aim for a globally optimal or close to an optimal motion that matches well the user-specified sketch, satisfies all user constraints and minimizes the objective function. In the experimental section we provide many examples that show that in contrast to locally optimal and greedy solutions, globally near-optimal solutions will not get stuck in a bad local minima and will avoid the dithering and inefficient patterns of motion that sub-optimal motions often have.

A number of on-line approaches have been created in the past few years that combine motion graph and interpolation techniques. Park and his colleagues [21, 20] 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 [13]. Shin and Oh [30] extended these techniques to include additional behaviors. They use a method similar to the one proposed by [7] to semi-automatically 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 [9] 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. In addition, our interpolated motion graph retains one of the main advantages of a motion graph – it automatically captures natural transitions between motion segments and does not require building a graph with contrived structure.

A number of sub-optimal algorithms have been developed to perform an off-line search of a motion graph quickly [12, 2, 24, 3, 5, 33, 18, 31]. To find an optimal solution efficiently, Lau and Kuffner [15] manually created an abstracted behavior-based motion graph with a very small number of nodes. In later work, they pre-computed search trees from the graph and used them for faster but not globally optimal search [14]. Lee and Lee [17] took the idea of pre-computation further by computing policies that, for each possible control input and avatar state, indicate how the avatar should move. Their approach allows interactive control with minimal run-time cost for a restricted set of control inputs. We also use precomputation to reduce the size of the graph significantly while retaining all unique functionality in the paths through the graphs.

Unlike all previous motion graph approaches with the exception of Lau and Kuffner [15](who search a small behaviour-based graph), we find a globally optimal solution rather than a locally optimal one. In Section 6, we show a number of comparisons to demonstrate that globally opti-

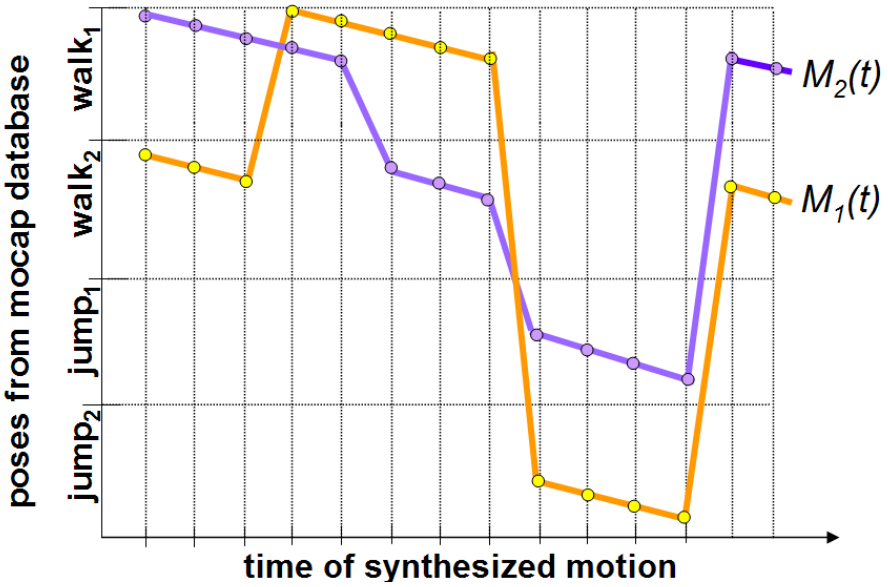


Figure 3: For this example, the database consists of five motions: three walks and two jumps. The resulting motion is an interpolation of two paths through the motion graph, $M_1(t)$ (orange) and $M_2(t)$ (purple). $M_1(t)$ and $M_2(t)$ can transition independently between motions.

mal solutions avoid the inefficient patterns of motion that are often seen with sub-optimal search techniques.

3 Overview

We construct a motion graph that supports interpolation and use A^* search to find an optimal path that satisfies the constraints specified by the user. In our representation, each pose of the resulting motion, $M'(t)$, is represented as an interpolation of two poses from the motion capture database:

$$M'(t) = wM_1(t) + (1 - w)M_2(t). \tag{1}$$

where $M_1(t)$ and $M_2(t)$ are poses from a motion capture database, w is the interpolation weight, and each M_i is a path through the motion graph. The two paths independently transition between motions in the database (Figure 3). We allow the segments of each path between contact changes to be independently scaled in time to synchronize the motions for interpolation. Similarly, w can be adjusted by the optimizer for each interpolated segment of motion.

We assume that a user provides a sketch of the $2D$ path that the character should follow in the environment and a set of constraints such as picking up an object, jumping, or walking to a specified location (Figure 4).

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

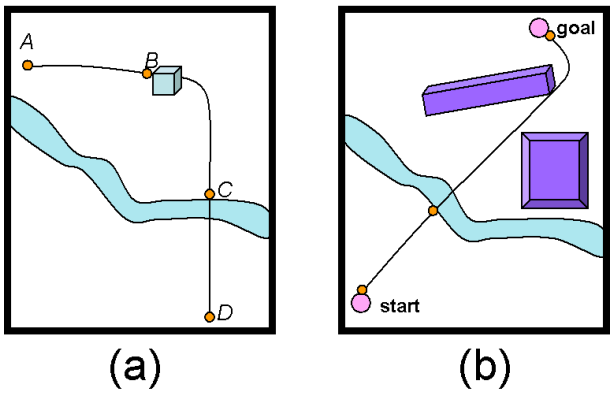


Figure 4: (a) User sketch for a motion with four constraints: start at position A, pick up object from table at position B, jump over a river at position C, and end at position D. (b) User specified only start and end positions. The system automatically creates a sketch of the $2D$ path from the start to the goal while avoiding obstacles.

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) ΔP_x and ΔP_z, the relative position of the root on ground plane (computed with respect to the previous pose in this motion sequence) and (4) ΔQ_{yaw}, the relative rotation of the root around the vertical axis (computed similarly).

Unknowns: The unknowns of the optimization problem are the variables from equation 1: $M_1(t)$, $M_2(t)$, and w. The weight, w is discretized and can take on 10 values.

Constraints: The constraints are the user-specified $2D$ path, other user constraints and environmental constraints such as walls and ditches. When the user specifies a path, the root of the character is constrained to stay inside a $2D$ corridor around that path ($0.5\,\mathrm{m}$ in most of the examples reported here). Small tolerances are also allowed for the other user-defined constraints.

Objective function: The objective function is a weighted average of two terms: a minimization of the sum of the squared torques computed via inverse dynamics and of the sum of the costs of transitions associated with edges in the motion graph. The first term is an approximation of the energy needed to compute the motion and the second is a measure of the smoothness of the motion.

Search: We use A^* search [22], and in particular its anytime extension ARA^* [19], to find the paths and interpolation weights that will satisfy the constraints and result in the optimal motion. The algorithm takes a graph as input, where each edge has a strictly positive cost, a start state, s_{start}, and a goal state, s_{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_{goal}. To guarantee the optimality of the solution and 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, $h(s) \leq c(s, s') + h(s')$ and for $s = s_{\text{goal}}$,

$h(s) = 0$, where $c(s, s')$ is the cost of a transition between states s and s'. 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 [27].

The anytime extension of A^*, ARA^* search [19], trades solution quality for search time by using an inflated heuristic (h-values multiplied by $\epsilon > 1$). The inflated heuristic often results in a speedup of several orders of magnitude. The solution is no longer optimal but is bounded by ϵ times the cost of an optimal solution. ARA^* starts by finding a suboptimal solution quickly using a large ϵ and then gradually decreases ϵ (reusing previous search results) until it runs out of time or finds a provably optimal solution.

4 Graph Construction

We first construct a standard motion graph, MG, from the motions in the database using techniques similar to those in the literature [18, 12, 2, 16]. We then construct a motion graph that supports interpolation which we call IMG—interpolated motion graph. We next construct a full search graph, ISG, by unrolling graph IMG into the environment. During the unrolling step, each state in graph IMG is augmented with the global position and orientation of the root causing the size of the graph to grow by the number of possible positions and orientations of the character in the environment. This step is needed so that we can search for motions that satisfy user-specified global position constraints and avoid obstacles.

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 in the motions and connecting them with edges as was done by Lee and his colleagues in [16]. They only allowed transitions when the character's contact with the environment changed. This decision drastically reduces the number of transitions and the size of the resulting graph but also removes many good transitions. We allow transitions inside of contact phases but prune redundant or non-optimal transitions as a pre-processing step (described in Section 5.1).

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. We want to 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.

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 transitions is much smaller as not all pairs of transitions in MG are feasible in IMG.

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 char-

acter about the vertical axis. The global variables are computed by interpolating and integrating the relative terms stored in graph IMG for the current state.

Graphs MG and IMG can be constructed as a preprocessing step. The unrolled graph, ISG is very large because each pose now appears many times in the graph, once for each reachable position and orientation. This graph, therefore, is constructed at a runtime on an as needed basis. In ISG, each transition is associated with a cost. This local cost is computed from the objective function.

5 Reducing the search space

Although graph ISG is much smaller than the full search space for a 60 dof character, it is still very large and cannot be searched at interactive frame rates. Each state in the ISG graph is defined by $S = (I_1, I_2, w, P_x, P_z, Q_{yaw})$. If we have a database with $N = 10000$ poses and we discretize w into 10 values, P_x and P_z into 1000 by 1000 values and Q_{yaw} into 100 values, we will have $NumStates = 10000^2 * 1000^2 * 100 * 10 = 10^{17}$ possible states. Because we constrain the character to stay inside the corridor around the user-specified path, only an estimated 1% of the position/orientation values are possible and $NumStates = 10^{15}$. The complexity of A^* algorithm is $O(E + SlogS)$, where E is the number of edges in the graph. With $S = 10^{15}$, the graph cannot be searched at interactive frame rates. Without interpolation, the maximum number of states is $S = 10^{10}$ which is still too large to search for an optimal or close to optimal solution at interactive frame rates. As a result, all existing approaches in the literature either find a solution using a greedy approach with no guarantee on sub-optimality or search a manually constructed graph with a small number of states.

To address this problem, 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 by culling states and transitions that are redundant or clearly sub-optimal because those states will not appear in any optimal solution. 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 6, we show that the combination of these techniques makes it possible to find an optimal or a close to optimal solution for a reasonable size database at interactive frame rates.

5.1 Graph compression

We compress the motion graph in two steps. First, we cull states and transitions that are clearly sub-optimal. Second, we cull states and transitions that are redundant. Both of the steps are done on graph MG. After the compressed version of MG is obtained, the graph IMG is derived from it in the way we have described in section 4.

Culling sub-optimal states and transitions: To cull transitions in MG, 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 an object in hand). The key insight behind our algorithm for culling transitions is that although there are likely to be many paths that directly connect two contact change states, they all end with the character in exactly the same pose and with the same

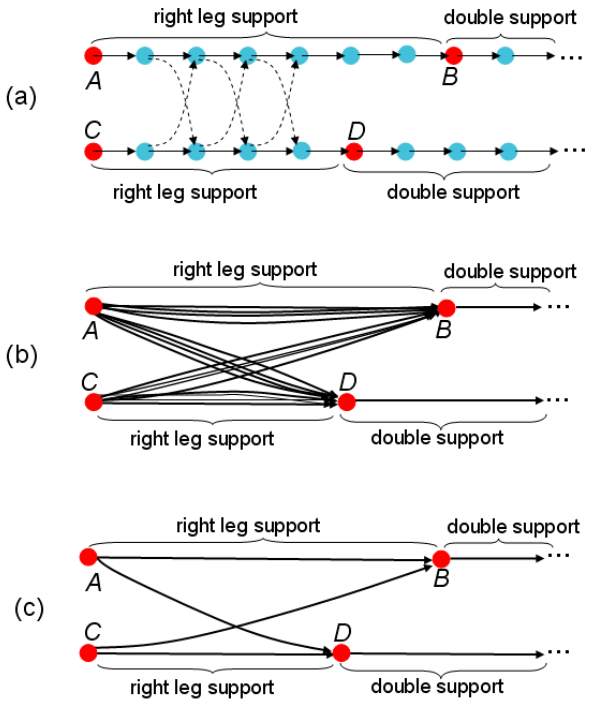


Figure 5: (a) States A, B, C and D (shown in red) are contact change states. If the character enters state A (and initiates a right leg support phase) it can exit only through state B or state D. (b) A representation with only the contact change states shows that there are many paths between each state. (c) The graph after transitions are culled to include only optimal paths between contact states.

root position and orientation (Figure 6). Therefore we can cull all but one of those paths without reducing the functionality of the graph. Figure 5 illustrates this process and we now explain it in more detail.

To determine the contact change states, we process all motions in the motion capture database by separating them into phases based on contact with environment. We use the technique of Lee and his colleagues [16] to detect contacts. The contact changes are then verified by a visual inspection of the motion and adjusted if necessary (only a very small percentage of the contacts need to be adjusted in locomotion and other simple behaviors).

Given the contact change states, we can compute the transitions between each pair of states. By definition, these transitions each contain exactly one change in contacts as the contact in the two states differ and all of the states that occur during the transition match the contact conditions of the first state. In general, there will be many transitions between any two contact change states but only one of these transitions is optimal with respect to our optimization function. The rest can be culled without affecting the optimality of the solution.

After culling the graph to include only optimal transitions between pairs of contact change states, the graph retains the same functionality in most situations. However, if the specification

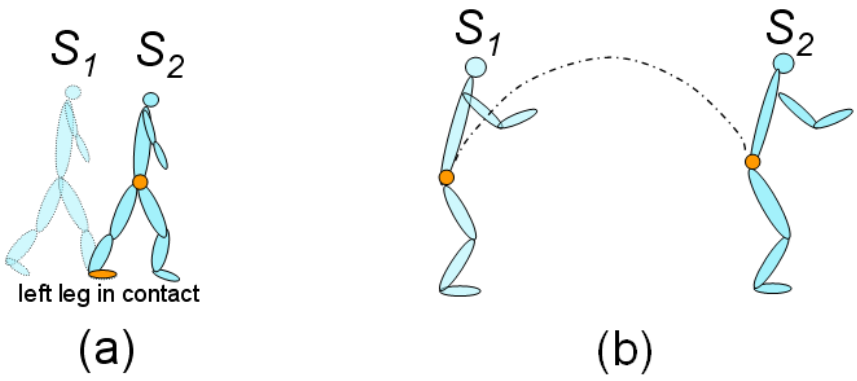


Figure 6: (a) For direct paths between a pair of contact change states S_1 and S_2, the global position and orientation of the root of the character at state S_2 is uniquely determined by the contact position and orientation at state S_1 and the values of the joint angles at state S_2. (b) The position 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.

provided by the user requires controlling the details of the motion during a period of time when the contacts are not changing, then the culled graph will have lost that functionality. For example, the user could no longer ask for waving while standing in place. Most constraints specify a change in contact (pick up the coffee cup, for example) and those constraints can be satisfied in the culled graph. Environmental constraints might be violated if the swept volume for the character from one contact change state to the next intersects an obstacle. This situation is uncommon because neither endpoint intersects the obstacle (or the search would not have explored the state) and for most human activities limited movement is possible with only one contact change.

The optimization function we use allows us to compute optimal paths as a pre-computation step because it is independent of the particular problem the user specifies. Many functions have this property: minimizing energy (while satisfying user constraints), minimizing sum of squared accelerations, maximizing smoothness, minimizing the total distance traversed from start to goal, minimizing total time of the motion, and satisfying specified annotations (for example behaviors or styles as in [3]). If the optimization function depends on what the user specifies, however, we then either need to have the user specification during pre-computation or approximate it with another function that is dependent on the user problem only at contact change states. For example, instead of minimizing the distance between every frame of the motion and the user-specified curve, we minimize the distance only between the contact change states and the curve.

Culling transitions in this way is not the same as just removing all the transitions except for the ones between contact change states, as others have done [16]. With that approach there would be no path between states A and D in Figure 5(a) even though one exists in the original motion graph. The preprocessing presented here retains many more transitions, a property that is important for

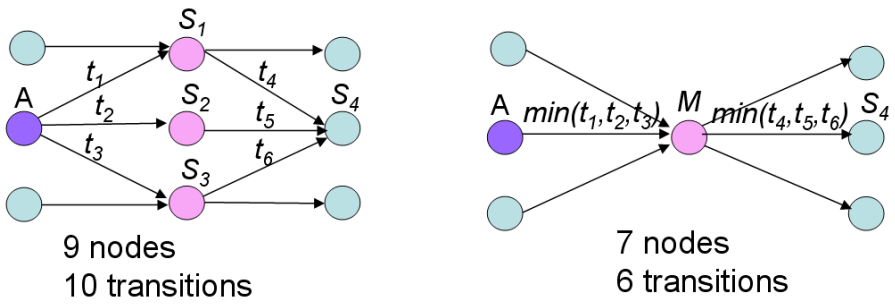


Figure 7: (a) States S_1, S_2 and S_3 are similar to each other. As a result, optimal transitions t_1, t_2 and t_3 are also very similar and all end with character at approximately the same position. (b) We can remove this redundancy if we merge states S_1, S_2 and S_3 into one state M and only keep the lowest cost transition.

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 the redundant states and transitions. This compression is therefore performed on the graph that contains only contact change states and the transitions are sequences of poses in between contact change states.

Motion graphs often include a lot of redundant data because of the need to capture natural transitions between behaviors. For example, to include natural transitions between walking and jumping, we included many steps of similar walking segments. As a result, each state in the motion graph may have many outgoing transitions that are very similar. We found that if we remove this redundancy we can drastically reduce the size of the graph.

For example, state A in Figure 7 has three successors: S_1, S_2 and S_3, that are similar to each other and all three transitions will end at the approximately the same position in the world when the graph is unrolled into the environment. We can cull the redundant states by merging all three states S_1, S_2 and S_3 into one and keeping only the lowest cost transition.

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. After we merge all possible states we will have redundant transitions between many of them. For example in Figure 7, transitions t_1, t_2 and t_3 become redundant after states S_1, S_2 and S_3 are merged. We keep just the lowest cost transition. We merge states in the order of their similarity: the most similar states are merged first and less similar states that still fall below the threshold for the similarity error are merged later. The similarity threshold for merging can be set substantially larger for merging states than for establishing the initial transitions. In merging, a higher threshold just removes flexibility from the graph whereas when transitions are created, it introduces perceptible errors. Because each transitions in the compressed graph is a sequence of poses representing one contact phase of the motion, we can post-process each transition to remove feet-sliding and other artifacts.

Constructing IMG from the compressed MG: Once the graph MG is compressed, the graph IMG is constructed from it as we have described in Section 4. In the compressed graph MG, how-

ever, 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. The interpolation uses a constant weight w throughout the transition. We use the interpolation scheme described in [28]. In particular, (1) during flight we interpolate the center of mass positions (instead of the root positions) and all the joint angles; (2) during ground contact we interpolate the positions of the feet, the center of mass positions and all non-redundant degrees of freedom to prevent the feet from sliding on the ground. In cases 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:

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

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

As was shown in [28], this interpolation scheme ensures that the majority of the created transitions are physically correct. In addition, we can also make sure that the transitions and any pair of consequent transitions are physically correct by just checking them at the time of construction.

5.2 Deriving informative lower bounds (heuristics)

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 A^* search explores depends on the quality of the heuristic function—the lower bounds on cost-to-goal values. Informative lower bounds can drastically reduce the search space exploration. In this section, we present a method for computing such bounds and in Section 6 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 needs to estimate the cost of getting to the goal for each state S in the graph ISG. We compute two heuristic functions, H_{pos} and H_{img}, and combine them to obtain one, informative heuristic function. The first heuristic function, H_{pos}, completely ignores the dynamics of the motion of the character which is encoded in the motion graph. For state S, $H_{pos}(S, G)$ estimates the cost of getting to the goal based only on the position of the character at state S and the position of obstacles in the environment. The second heuristic function, H_{img}, compensates the first function by taking into account the capabilities of the character that are encoded in the motion graph but ignores its position in the environment. We now describe how to compute H_{pos} and H_{img} and how to combine the two.

The heuristic function based on the character location in the world (H_{pos}): To compute $H_{pos}(S, G)$ we first compute the shortest distance from the location of the character at state S to the goal location, G. This distance is computed as the shortest path in $2D$ from S to G taking into account obstacles in the environment. We also require for this path to stay inside the corridor around the user specified path (Figure 8(a)). We use a single $2D$ Dijkstra's search to obtain such path from every cell to the goal right before the full search starts. The search takes only few milliseconds. In our implementation, when computing the H_{pos} function, the environment is discretized into 0.2 by 0.2 meter cells. During the full search, the root position of the character was

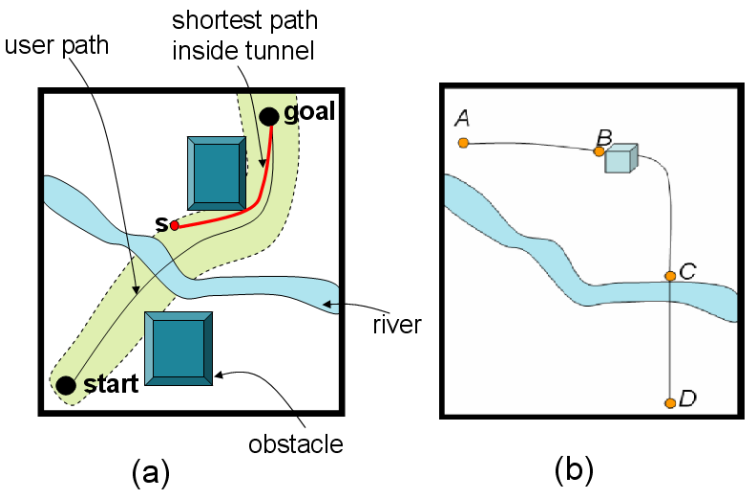


Figure 8: (a) $H_{\text{pos}}(S, G)$ is the shortest path from the position of the character at state S to the goal. The shortest path is constrained to stay inside the tunnel. (b) User sketch. The character starts at position A, picks up an object from a table at position B, jumps over a river at position C and arrives at position D.

discretized into 0.02 by 0.02 meter cells. Because our cost function minimizes the sum of squared torques, we need to multiply the shortest distance (which is in meters) by the minimum torque required to traverse one meter. We compute this minimum torque from the motion graph data.

The heuristic function based on motion graph state (H_{img}): $H_{pos}(S, G)$ usually provides a good estimate of the cost to the goal for motions that require character to travel from one location in the world to another and do not include any user or environmental constraints. User or environmental constraints usually require much more effort from the character to pass the constraint than the minimum torque estimate assumed by H_{pos}. Also, the motion graphs restrict what the character can do from a particular state. For example, if on the way from state S to the goal the character needs to jump across an obstacle and it is very difficult to reach a jumping motion from state S, then the cost-to-goal at state S should be very large. H_{pos} will grossly 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 various constraints for each state in the interpolated motion graph.

The user provides a sketch of the motion and a set of constraints. For example, the user may want the character to start from position A, pick up an object from a table at position B, jump over a river at position C and arrive at position D (Figure 8(b)). This motion has four constraints (three user and one environmental): (1) start at position A; (2) pick an object from the table at position B; (3) jump over river at position C and (4) arrive at position D. For each of these constraints, we can compute the minimum cost of passing it based on the available paths in the interpolated motion graph and use it as the heuristic H_{img}.

The general idea is that for each type of the constraint supported by our system, $H_{\text{img}}(S, C)$ is

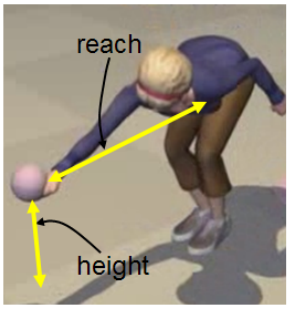
computed as the minimum cost of getting to the pose in the motion graph that satisfies the constraint C from the state S. The computation of the H_{img} heuristic does not depend on the particular user problem and therefore can be pre-computed off-line. Also, the computation of the H_{img} heuristic is completely automatic assuming that contact information for all motions in the motion capture database is available. Contact information can be computed automatically using one of the existing techniques (for example a work of Callennec and Boulic [4]).

For example, for the picking constraint C, $H_{\text{img}}(S, C)$ represents the minimal cost of a path in the interpolated motion graph from state S to a state that represents the picking up of an object. Each "picking" pose can be defined by two parameters: *height* and *reach* (Figure 9(a)). *Height* defines how high the object is located with respect to the ground. *Reach* defines how far the character reaches out to pick up the object (distance between the root and the hand projected onto ground). Based on the contact information, we automatically identify each state in the graph IMG that represents picking up an object, a "picking" state, $p = (I_1, I_2, w)$, where both poses, I_1 and I_2 are states where an object can be 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 a table (Figure 9(b)). Each cell in this table represents a range of *height* and *reach* values and the value in this cell is the minimal cost of getting from the given state to a "picking" state with *height* and *reach* values in this range. For each entry in the table and each state in IMG, we search graph IMG to compute the cost for that entry. IMG is quite small and therefore this pre-computation is reasonably fast.

The exact same method of computing $H_{\text{img}}(S, C)$ applies to many types of constraints. While in our current implementation we support five types of constraints including picking, jumping, stepping onto obstacles (beam for example), ducking and sitting, the method should generalize to other types of constraints such as for example kicking, stepping over obstacles or standing on one leg.

Combining the two heuristics: While $H_{\text{pos}}(S, G)$ is an estimate of the cost of getting to the goal, $H_{\text{img}}(S, C)$ is an estimate of the cost of satisfying a particular constraint, C. We combine these quantities into a single heuristic function by summing them together. This summation, however, may overestimate the actual cost to the goal, which would violate the required properties of the heuristics. This may happen because the $H_{\text{img}}(S, C)$ term also accounts for the motion of the character in the plane, and this is already accounted for in the $H_{\text{pos}}(S, G)$ term. To resolve this problem, when computing the $H_{\text{img}}(S, C)$ term the cost of each transition in the interpolated motion graph is reduced by the minimum torque required to traverse the planar distance covered by the transition. This is the same torque value as used for the computation of $H_{\text{pos}}(S, G)$. We denote this motion graph heuristic by $\tilde{H}_{\text{mg}}(S, C)$. If at state S there are still n constraints remain to be satisfied, we fetch $\tilde{H}_{\text{mg}}$ term for each of these constraints, and then add all of them to the $H_{\text{pos}}(S, G)$ term to obtain a full heuristic value for state S:

$$H(S) = H_{\text{2d}}(S, G) + \sum_{i=1...n} \tilde{H}_{\text{mg}}(S, C_i) \tag{3}$$



(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 9: (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 pre-compute this table. Each entry in the table represents the minimal cost of getting to a "picking" state where the height and reach parameters are within the given range.

6 Experimental results

To illustrate the effectiveness of our approach, we generated a variety of different 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 additional constraints such as picking an object or sitting on a chair. Based on the sketch and the current environment we automatically compute environmental constraints such as: stepping onto an obstacle, ducking and jumping. The motions illustrating the experiments are shown on the accompanying video.

An example shown in Figure 2 shows a motion of character getting over an obstacle course. The character walks over the beam, jumps over holes, ducks under a bar and finally sits on the chair. This example illustrates that our algorithm can synthesize motions that are 15 seconds long and consist of many different behaviors. Besides obstacle course examples, we have also synthesized many other examples, including walking along curves of different curvature, picking and placing an object in various locations, jumping over stones with variable spacing, forward jumps of different lengths, vertical jumps with different amounts of rotation and forward walks of different step lengths. Figure 10 shows images for some of the results. It took from a few seconds to 3 minutes to compute a close to optimal solutions for all examples. For smaller, single behavior examples, such as jumps and short walks, it takes from few milliseconds to few seconds to compute an optimal solution. For longer, multi-behavior examples it takes from few second to 3 minutes to compute a close to optimal solution. In general, the time depends on the size of the database used to construct the interpolated motion graph, the length of the generated motion and the complexity of the constraints. The first, sub-optimal solution is usually found in milliseconds for short examples (such as forward jumps) to a a few seconds for more complex examples. We next

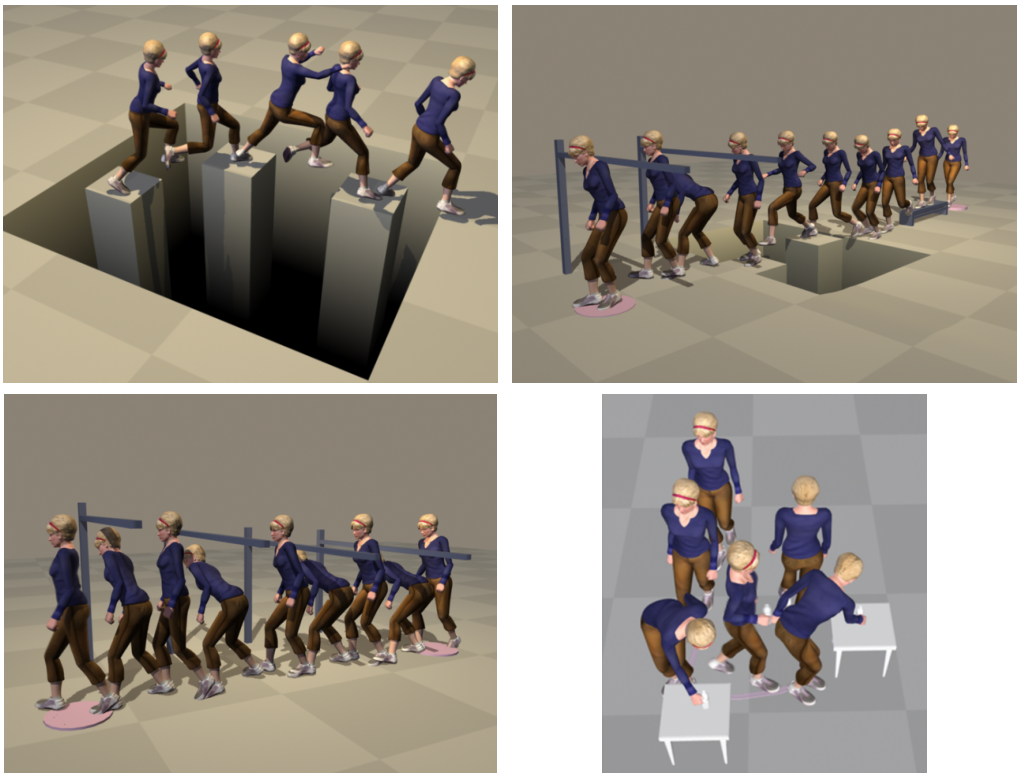


Figure 10: Synthesized motion

describe a number of experiments we performed to better assess the performance of our algorithm.

6.1 The benefit of interpolation

The first experiment shows why interpolation in conjunction with motion graphs allows us to satisfy user-specified constraints within a small error tolerance. We used the following three test problems: (1) a character needs to start at position A and pick up a small object at position B (see Figure 11a); (2) a character needs to start at position A and pick up a small object at position B but now we also constrain the root position of the character to position C while picking up a small object (see Figure 11b); (3) a character needs to start at position A and walk to position B with one walk cycle (see Figure 11c).

We ran many experiments for each problem by sampling the locations of constraint B (179 samples for first two problems and 10 samples for third problem). Tables 1 summarize the results for each problem for interpolation and no interpolation. The results show that with interpolation, the system consistently finds solutions for small error tolerances. Without interpolation, we need to set the error tolerances much higher to get consistent results.

We also computed the average cost (sum of squared torques) of the solutions for the second problem. Only experiments that succeeded were included in the computation of the average cost. The average solution cost is about 1.35 times higher without interpolation, especially for small

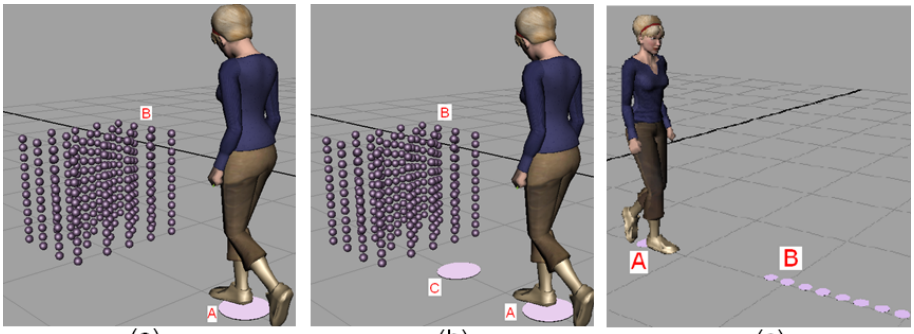


Figure 11: Test problems: (a) A character needs to start at position A and pick up a small object at position B. We sample the location of constraint B; (b) A character needs to start at position A and pick up a small object at position B but now we also constrain the root position of the character to position C while picking a small object. We sample the location of constraint B; (c) A character needs to start at position A and walk to position B with one walk cycle. We sample the location of constraint B.

tolerances. Without interpolation the search has much less freedom to satisfy the constraints. Therefore even if the search is able to find the solution, this solution is more likely to contain dithering and inefficient motion patterns that result in the higher solution cost (the movie shows a few examples).

The previous examples demonstrated that interpolation is often required to satisfy the constraints. We now demonstrate that selecting two paths for interpolation simultaneously 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. Interpolation usually produces solutions with better, more optimal strategies and is also more robust (do not need to tweak the threshold as will be explained next).

The example in Figure 12 shows a situation where the solution that satisfies all user-specified constraints does exist even without interpolation. In this example the character needs to get over three columns. The strategy found with interpolation is more optimal than the one without. With interpolation, the character walks from the first to the second column and jumps from the second to the third column. Without interpolation, on the other hand, the character jumps between the columns. Because the optimal strategy without interpolation is different than the one with interpolation it would be impossible to warp one solution into the other.

The example in Figure 13 shows a situation where the solution that satisfies all user-specified constraints within the user-specified tolerance does not exist without interpolation. In this example the character needs to follow the curve within the user-specified tolerance (represented as a corridor around the curve). With interpolation, the solution exists and the character can track the curve very well. Because the solution does not exist without interpolation we need to increase the corridor width around the curve to find a solution. In the resulting solution, however, the character takes only five steps, whereas with interpolation she takes six steps. It is therefore unclear how

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

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

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

Table 1: The success rate for each of the three test problems for three different error tolerances and interpolation/no interpolation. Pick up object (top): we have uniformly sampled 179 locations in $3D$ for the object (Figure 11a). Pick up object from a given location (middle): now the character must stand in a specified location when picking up the object (Figure 11b). Walk to goal location: uniform sampling of goal location along the X axis (Figure 11c) resulting in walks with different step lengths. An experiment is counted as a success if the search finds a solution within two minutes and within a specified error tolerance.

to warp this solution to satisfy the tolerance constraints. As in the previous example, this is a different strategy for the solution and it would not be possible to warp the solution found without interpolation into the solution found with interpolation.

It is also often hard to decide by how much to increase the tolerance. Increasing tolerance too much may result in a solution that would be impossible to warp. For example, we often cannot find any solution without interpolation for jumping across arbitrary placed columns. If we increase the width of the column too much the character walks across the columns instead of jumping but to satisfy the original column width the character actually needs to jump. In this case the algorithm can not find any solution at all without interpolation because it is impossible to warp a walk into a jump.

In general, often there is no solution without interpolation that satisfies user constraints within the user-specified tolerance. Increasing the tolerance allows a solution to be found. But then, the found solution often follows a different, less optimal strategy than the solution with interpolation and therefore cannot be warped to match an optimal strategy and often can not even be warped to satisfy the user and environmental constraints.

6.2 The benefit of optimality

In the accompanying movie we show many examples that illustrate that globally near-optimal solutions and avoid the dithering and inefficient patterns of motion that sub-optimal solutions often have. The solution cost largely decreases as its optimality increases during our search. Tables 2 show the results for three motions: a walk from start to goal; a walk from start to goal that requires crossing three rivers of different width; a walk to a place where the character needs to pick up an object. As the optimality of the solution increases the character finds more efficient motion

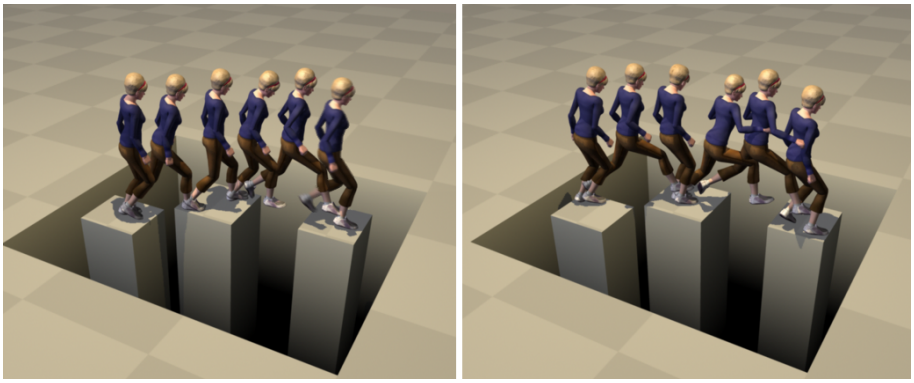


Figure 12: (Left Image) The optimal solution with interpolation. The character walks across the first column and jumps across the second column. (Right Image) The optimal solution without interpolation. The character jumps between each pair of columns. Because the behaviors change between the two solutions, we cannot warp the solution found without interpolation to match the more efficient solution found with interpolation.

patterns (movie shows many more examples).

6.3 The benefit of motion graph compression

In this experiment, we evaluate the effect of the motion graph compression. Table 3 shows general statistics for three different databases: (1) walking, jumping, ducking, sitting and walking along the beam motions; (2) walking and picking motions; (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 (merging transitions) and after the second compression step (merging states). The table also gives the time required to compress the graph (a pre-computation step performed only once for each database). Compression techniques reduce the size of the graph by a factor of 20 to 50.

6.4 The benefit of the heuristic function

We also evaluated the effectiveness of our heuristic function. The results are shown in Table 4. We compare four heuristics: (1) the Euclidean distance to the goal; (2) only the H_{2D} component of our objective function; (3) only H_{mg} component of our objective function; (4) the combined heuristic function with both H_{2D} and H_{mg} components. The results demonstrate that our heuristic function is essential for making 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 for getting good results—leaving just one component and disabling the other makes the search substantially less efficient.

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
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 (ϵ)
0.016	10,700,000	10.0
0.032	8,200,000	2.0
0.844	6,250,000	1.0

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
0.70	1,200,000	10.0
9.10	650,000	2.0
20.10	550,000	1.0

Table 2: Top table (motion 1): walk from start to goal; the first solution is visually very suboptimal—the character takes two awkwardly large steps to reach the goal position (leftmost image in Figure 1); the second solution is better—the character makes smaller steps but the walk is a bit unnatural because the steps are of different length; the final solution is optimal and looks natural (second image in Figure 1). Middle table (motion 2): walk with jumps over three rivers; the first solution is suboptimal—the character takes two legged jumps to cross all three rivers which is quite inefficient; the second solution is more natural, the character now uses a one-legged jump; in the optimal solution the character steps over the last and smallest river. Bottom table (motion 3): the character starts on the small rectangle and walks to a small object; the first solution requires an unnatural posture (third image in Figure 1); the second solution is better but the character reaches in from the side to contact the object; the optimal solution looks natural (rightmost image in Figure 1).

	Before Merging	After merging transitions	After merging states	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 3: Compression for three motion graphs. The first graph is computed from walking and jumping motions. The second graph is computed from walking and picking motions and the third one is computed from just walking motions.

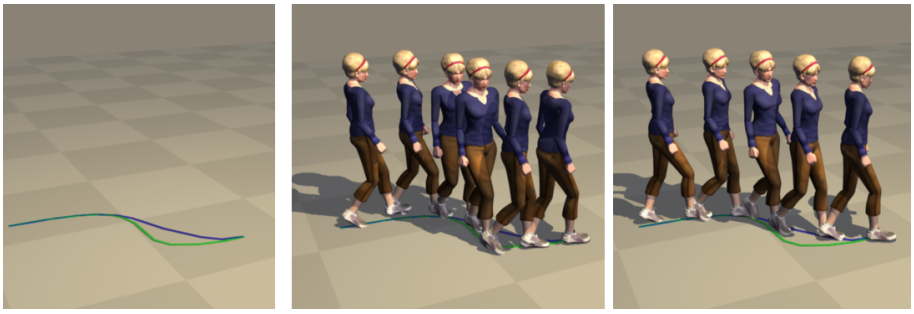


Figure 13: (Left Image) The user wants the character to walk along green curve. The blue curve shows the trajectory of the root for the best solution without interpolation (Middle Image) The solution with interpolation. The character can track the curve, deviating only slightly from the prescribed path. (Right Image) 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. It is therefore impossible to warp the solution without interpolation into the one found with interpolation.

ε	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 4: Evaluation of the heuristic function. Each column shows the runtime of the search in seconds and the number of states expanded by it for different heuristic functions. The first column is for the Euclidean distance to the goal. The second column uses only the H_{2D} component of our objective function. The third column uses only H_{mg} component of our objective function. The last column shows the results for the combined heuristic function. The first row shows search efforts to obtain a solution whose cost is at most 10 times the optimal one. The sub-optimality bound for the second row is 3. The solution in the last row is optimal.

7 Discussion

In this paper, we built a discrete reduced-space representation of human motion. This representation can be viewed as a combination of a motion graph and interpolation techniques. The motion that can be generated with this representation is an interpolation of two time-scaled paths through the motion graph. Finding a solution in this smaller search space is much easier than finding a solution in the full search space. In addition, the synthesized motion is likely to contain natural coordination patterns. This objective is difficult to describe mathematically and is therefore hard to achieve when searching the full search space. We have shown that the optimization in this discrete space supports interactive frame rates and allows for the synthesis of less dynamic motions and longer motions that are composed of different behaviors. It takes less than 3 minutes to compute a close to optimal 15 second long motion and the first sub-optimal solution is usually found in just a few seconds.

We compute motions that minimize the sum of squared torques as an objective function while satisfying user constraints. Our objective function together with the requirement that we interpolate motion segments with the same contact seems to work well to pick and synchronize motion segments that when interpolated result in natural motions. For example, two walk segments with both arms to the side are more likely to be picked by the search for interpolation than segments with one arm waving because the latter requires more energy (unless the constraints specifically require waving).

In this paper we show that finding close to an optimal solution is often important for obtaining good quality motion. To find optimal motion we develop motion graph compression techniques and derive informative heuristic function. While we used these methods to find optimal solutions in interpolated motion graphs, they can also be used to efficiently find optimal solutions in the regular motion graphs.

The quality of the results largely depend on the quality of the motion database used to construct a 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 give us the desired motion.

We also found that it is important that the resulting motion graph has "good" connectivity. For example, if it is impossible to reach a "picking" state from other states in the motion graph then we cannot synthesize motions that satisfy constraints that require picking up objects. Our experiments show that to obtain good results it is important that many states in the motion graph can quickly reach a constraint state (such as a "picking" state) and many states can be quickly reached from constraint states. An automatic technique for evaluating the connectivity of a motion graph would be very helpful. In their work, Reitsma and Pollard [25] 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. 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 12,000 frames (motions are sampled at 30 frames a second). Scaling our approach to the larger databases, however, will require additional work. We plan to experiment with automatic clustering of motions into behaviors and only interpolating motions within the same class to further reduce the size of the problem.

All of our results are for no interpolation or interpolation of two paths. In all of our experiments, interpolation of two paths was sufficient to find a solution that met the users constraints and produced natural looking motion. Some problems, however, may require interpolation of more than two paths in order to satisfy the constraints. We would like to scale our approach to interpolation of three paths. If this proves to be infeasible, we can also try an iterative approach: first compute the best possible solution for interpolation of two paths; then compute a second solution for the interpolation of two paths, which, 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 interpolation of two paths is sufficient to identify the right sequence of behaviors (strategy) then small refinements should work well.

Our approach can currently find a close to an optimal solution for motions that are approximately 10 seconds long. Scaling our approach to longer motions is part of the future work. As the length of the desired motion increases, the complexity of the search greatly increases. We can decrease the complexity of the search by limiting the number of contact phases in the final solution that involve interpolation. Interpolation is most often required whenever the motion needs to satisfy user-specified constraints (such as the position for picking up the object or curvature) and is generally not necessary when the character is moving in free space. Deciding in advance when to allow interpolation and when not to would be difficult because the steps before the constrain are likely to require interpolation. We can limit interpolation by adding one more variable to each state in the graph that counts the number of motion segments that have been interpolated so far. We can then limit this variable during the search and therefore control the maximum number of segments interpolated in the solution. We have run preliminary experiments with this approach and have found that interpolating only when it is a real benefit can greatly reduce the complexity of the search.

References

[1] Yeuhi Abe, C. Karen Liu, and Zoran Popović. Momentum-based parameterization of dynamic character motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 173–182, 2004.

[2] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, July 2002.

[3] Okan Arikan, David A. Forsyth, and James F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3), 2003.

[4] Benot Le Callennec and Ronan Boulic. Robust kinematic constraint detection for motion data. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2006.

[5] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2):182–203, 2003.

[6] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22(3):417–426, 2003.

[7] M. Gleicher, H. Shin, L. Kovar, and A. Jepsen. Snap-together motion: Assembling runtime animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[8] S. Guo and J. Roberge. 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*, 1996.

[9] Rachel Heck and Michael Gleicher. Parametric motion graphs. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation (Poster Presentation)*, 2006.

[10] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 214–224, August 2003.

[11] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics*, 23(3):559–568, August 2004.

[12] Lucas Kovar, Michael Gleicher, and Fred Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, 2002.

[13] Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 29–38, July 2005.

[14] Manfred Lau and James Kuffner. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, September 2006.

[15] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 271–280, 2005.

[16] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500, 2002.

[17] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 79–87, 2004.

[18] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. *ACM Transactions on Graphics*, 21(3):465–472, 2002.

[19] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems (NIPS) 16*. Cambridge, MA: MIT Press, 2003.

[20] Sang Il Park, Hyun Joon Shin, Tae Hoon Kim, and Sung Yong Shin. On-line motion blending for real-time locomotion generation. *Computer Animation and Virtual Worlds*, 15(3-4):125–138, 2004.

[21] Sang Il Park, Hyun Joon Shin, and Sung Yong Shin. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 105–112, July 2002.

[22] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[23] Ken Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[24] Katherine Pullen and Christoph Bregler. Motion capture assisted animation: texturing and synthesis. *ACM Transactions on Graphics*, 22(2):501–508, 2002.

[25] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character navigation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–98, 2004.

[26] Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications*, 18(5):32–40, September 1998.

[27] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 2003.

[28] Alla Safonova and Jessica K. Hodgins. Analyzing the physical correctness of interpolated human motion. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 171–180, 2005.

[29] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics*, 23(3):514–521, 2004.

[30] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: Constructing an interactive character with continuous controls. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, September 2006.

[31] Madhusudhanan Srinivasan, Ronald A. Metoyer, and Eric N. Mortensen. Controllable real-time locomotion using mobility maps. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 51–59, 2005.

[32] Adnan Sulejmanpašić and Jovan Popović. Adaptation of performed ballistic motion. *ACM Transactions on Graphics*, 24(1):165–179, 2005.

[33] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 291–300, July 2005.

[34] Douglas J. Wiley and James K. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics Applications*, 17(6):39–45, 1997.

[35] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics (Proceedings of SIGGRAPH 88)*, 22(4):159–168, August 1988.