

Synthesizing Human Motion From Intuitive Constraints

Alla Safonova¹ and Jessica K. Hodgins²

¹University Of Pennsylvania, USA
alla@cis.upenn.edu

²Carnegie Mellon University, USA
jkh@cs.cmu.edu

June 10, 2008

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. Motion graphs and their variations have proven to be a powerful tool for synthesizing human motion when only a rough sketch is given. Motion graphs are simple to implement, and the synthesis can be fully automatic. When unrolled into the environment, motion graphs, however, grow drastically in size. The major challenge is then searching these large graphs for motions that satisfy user constraints. A number of sub-optimal algorithms that do not provide guarantees on the optimality of the solution have been proposed. In this paper, we argue that in many situations to get natural results an optimal or nearly-optimal search is required. We show how to use the well-known A* search to find solutions that are optimal or of bounded sub-optimality. We achieve this goal for large motion graphs by performing a lossless compression of the motion graph and implementing a heuristic function that significantly accelerates the search for the domain of human motion. We demonstrate the power of this approach by synthesizing optimal or near optimal motions that include a variety of behaviors in a single motion. These experiments show that motions become more natural as the optimality improves.

Keywords: computer graphics, animation, planning, motion capture, human motion, motion graphs

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 1). Optimization is a common technique for finding a motion when only a rough sketch is provided. A number of continuous optimization techniques have been proposed for solving this problem (for example, [1–4]). In this paper, we concentrate on discrete optimization techniques that search a graph constructed from existing motion capture data for the solution.

Motion graphs have proven to be a powerful technique to solve for a desired motion based only on a rough sketch [5–9]. Because the solution is constrained to a sequence of motion segments from the motion capture database, motion graphs are restrictive in the set of motions they can represent. For example, it would be impossible to synthesize a motion for picking up a cup from a table that is 1.0 meter high if the database contains only motions for picking up a cup from tables that are 0.5 and 1.5 meters high. To relax this restriction, in [10] we have introduced interpolated motion graphs (IMG). The motion is represented 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. Although larger than a motion graph, 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.

Discrete search techniques can be used to search a motion graph or an interpolated motion graph for a motion that satisfies user-specified constraints. During the search, the graph is unrolled into the environment (by 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 causes the size of the graph to grow drastically in size and makes search challenging.

Most motion graph implementations have used sub-optimal techniques (that do not provide guarantees on the optimality of the solution) because it was thought

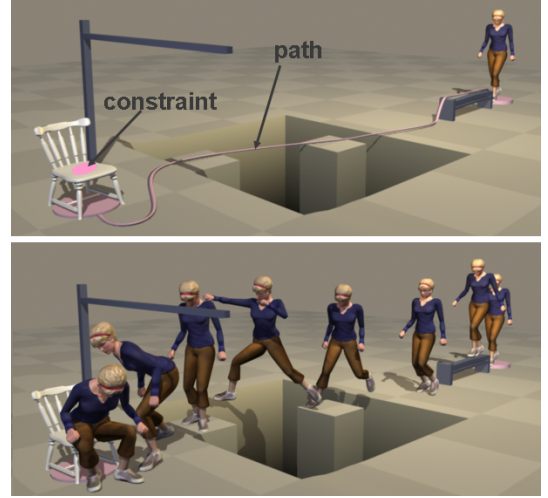


Figure 1: (Left) Rough sketch of the desired path and a user constraint requiring the character to sit on the chair at the end of the path. (Right) Synthesized motion.

to be infeasible to perform a global search of a motion graph of sufficient size to produce natural motion. In this paper, we argue that in many situations to get natural results an optimal or nearly-optimal search is required.

We show how to use the well-known A^* search (we use an anytime version of A^* by Likhachev et. al [11]) to find solutions that are optimal or of bounded sub-optimality in a motion graph containing a variety of behaviors. A^* is a breadth-first algorithm that gains its efficiency by using a lower bound on the cost to the goal (a heuristic) to avoid computing nodes that are irrelevant to the optimal plan.

We were able to use A^* on a motion graph because we made two improvements to the standard motion graph implementation. 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 optimal solution or are redundant. The compression reduces the size of the graph by a factor of 20 to 50.

The second technique computes an informative heuristic function that guides the search toward states that are more likely to appear in an optimal solution. We created the heuristic by splitting the full search problem into two much simpler problems: one that is only concerned with the character’s location and orientation in the environment and the other based purely on the motion graph with no environmental constraints. Both of these smaller planning problems can be solved efficiently. The solutions to these problems provide lower bounds on the costs to the goal that can be combined to obtain an informative heuristic function for the search.

The combination of these two techniques makes it

possible to find optimal or close-to-optimal solutions in standard motion graphs at close to interactive rates (10 seconds of motion required a few seconds of computation time for our examples). It also makes it possible to search interpolated motion graphs for an up to 15 second motion with a few minutes of computation.

In our experiments the user specified a desired 2D path that required the character to perform various behaviors such as jumping, walking, walking along a beam, ducking, picking, sitting. The solution minimizes the sum of squared accelerations (an approximation to energy) and therefore our implementation avoids the dithering back and forth motions often seen in solutions computed with a local or sub-optimal search [7]. The optimality of A^* allows us to find motions which are natural in that they use a running jump for longer jumps and a standing broad jump for shorter jumps as a human likely would.

2 Background

Continuous optimization, introduced to the graphics community by Witkin and Kass [1], 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 [2–4]). 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.

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) [7] and *off-line* approaches where the full motion specification is known in advance [6, 8]. 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. Kovar and his colleagues [6] employed a branch and bound algorithm to get an avatar to follow a sketched path. Arikan and Forsyth [8] created a hierarchy of graphs and employed a randomized search algorithm for the synthesis of a new motion subject to user-specified constraints. Pullen and Bregler [12] segmented motion data into small pieces and rearranged them to match user-specified keyframes. In 2003, Arikan and his colleagues [9] presented a new

search approach based on dynamic programming that supports user-specified annotations of the motion. The search space for their algorithm is much smaller than for ours because they do not include position information with each state but only the character’s pose and a time. This simplification makes it difficult to satisfy position constraints. Choi and his colleagues [13] presented a scheme for planning natural-looking locomotion of a biped figure based on a combination of probabilistic path planning and hierarchical displacement mapping. Sung and his colleagues [14] used probabilistic roadmaps and displacement mapping to synthesize motion for crowds.

To guarantee fast performance none of these approaches find optimal solutions but instead use sub-optimal search techniques. To find an optimal solution efficiently, Lau and Kuffner [15] 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 [16]. Lee and Lee [17] 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.

Unlike all previous motion graph approaches with the exception of Lau and Kuffner [15], we find a globally optimal or a close-to-optimal solution with an upper bound on the sub-optimality. In Section 6, 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 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) ΔP_x and ΔP_z , the relative position of the root on the 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).

The goal is to find motion that satisfies user sketch and constraints and at the same time is natural and physically correct. Optimization is a common technique for finding a motion when only a rough sketch is provided. The user specifies a set of constraints (such as pose) and an objective function. The optimization problem is

then to minimize the objective function while satisfying user-specified and physics constraints (which preserve the physical validity of the motion). In this paper we concentrate on discrete optimization techniques.

To setup optimization problem we need to define unknown variables that optimizer will search for, constraints that optimizer will need to satisfy and the objective function that optimizer will minimize to find a natural solution. In Section 4 we describe each of them.

To solve the optimization problem we use a database of motions to construct a graph that will be searched for an optimal solution. A number of different graphs have been proposed in the literature, including behavior-based graphs, motion graphs and graphs that combine motion graphs with interpolation. We describe graph construction process in Section 5.

A number of search algorithms can be used to search constructed graph for a solution. Global search methods compute the whole solution in one search. Local search methods, on the other hand, only perform short horizon searches that repeatedly find few steps of the solution. Global methods find better quality solutions at the expense of longer computation times. Global search methods can be divided into methods that find a globally optimal solution and ones that find a locally optimal solution with no guarantee on sub-optimality. We discuss the importance of finding an optimal solution in Section 6.

In Section 7 we show how to use the well-known A^* search (we use an anytime version of A^* [11]) to find solutions that are optimal or of bounded sub-optimality. We achieve this goal for large graphs by performing a lossless compression of the graph and implementing a heuristic function that significantly accelerates the search for the domain of human motion. We present our results in Section 8 and conclude with the discussion in Section 9.

4 Discrete Optimization Problem

Unknowns: Given user sketch and constraints (such as one shown in Figure 1), the goal is to find the motion of the character that stays close to user sketch and satisfies all constraints. Therefore, unknowns of the optimization problem are poses of the character over time.

Objective function: Variety of objective functions have been proposed in the literature. For example, in their work [8], Arikan and Forsyth, use an objective function that is a summation of violations of soft constraints and smoothness of the motion. Soft constraints include: position and orientation constraints, number of frames in the resulting motion, joint constraints, style constraints and so on. Kovar and his colleges in [6], find a motion of the character that follows a sketch of the path provided

by the user. The objective function they use is the difference between a user given path P and the actual path traveled P traversed by the character. In [9], Arikan and his colleges find a motion that satisfies given user annotations. Their objective function is a summation of D and C functions, where D measures the difference in annotations for each frame of the motion and C measures the smoothness (“distance” between features of 2 consecutive frames).

In our work, 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 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 that will result in efficient motion patterns. The second term is a measure of the smoothness of the motion.

Constraints: Constraints allow user to express goals that need to be satisfied exactly or within small tolerance. When constraints do not need to be satisfied exactly but as close as possible, then they are treated as soft constraints and added as a term to the optimization function (as was done in [9]).

Hard constraints are most often used to specify start and goal locations of the character as was for example done in [8, 9]. Obstacle avoidance constraints are also treated as hard constraints.

In our work, all 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 7.3). User can, for example, specify a set of constraints that constraint a particular point on the character body (such as sitting on a chair or picking an object). A user also either 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 2). The root of the character is constrained to stay inside a 2D corridor around the path (0.5 – 1.0 m wide in most of the examples reported here). If the user sketch passes across obstacles (such as a river) the system also automatically adds environmental constraints (which are used for the computation of heuristic function that guides the search). Finally, obstacle avoidance constraints are automatically included. Figure 2 gives an example of user sketch. User constraints should coincide with contact changes in the motion and will be met within a small tolerance.

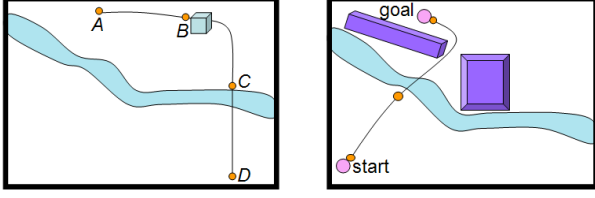


Figure 2: Two example problem specifications. (Left) The user provided the sketch of the path of the character and specified 3 constraints: start at A, pick an object from a table at B, and arrive at D. An environmental constraint for jumping over the river is added automatically by the system. (Right) 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.

5 Graph Construction

To solve the optimization problem we use a database of motions to construct a graph that will be searched for an optimal solution. A number of different graphs have been proposed in the literature, including behavior-based graphs, motion graphs and graphs that combine motion graphs with interpolation. In this work we show how to search standard motion graphs and interpolated motion graphs for near-optimal solutions. We give an overview of the graph construction process for these graphs next.

5.1 Motion Graphs

Motion graphs are unstructured graphs which are created completely automatically. A motion graph, MG , is constructed by finding “similar” poses in different motions and creating transitions between these poses (Figure 3). A motion can be generated by simply traversing a path in the graph. Two states are considered similar if the error between them is within some threshold. Different error functions have been shown to work (see for example [6–8]).

5.2 Interpolated Motion Graphs

Motion graphs capture natural transitions between available motion and as a result allow for creation of long, multi-behavior motions. Because the solution is constrained to a sequence of motion segments from the motion capture database motion graphs can not synthesize variations. For example, it would be impossible to synthesize a motion for picking up a cup from a table that is 1.0 meter high if the database contains only motions for

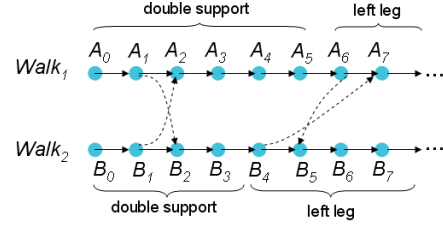


Figure 3: A simple motion graph for two walking motions. States A_1 and B_1 are similar and therefore two transitions are added to the motion graph: $A_1 \rightarrow B_2$ and $B_1 \rightarrow A_2$.

picking up a cup from tables that are 0.5 and 1.5 meters high.

To relax this restriction, we have introduced interpolated motion graph, IMG ([10]). The key insight behind interpolated motion graphs is that the motion is represented 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 [18]. Although larger than a motion graph, 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 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 4). 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 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. We first construct a standard motion graph, MG as described in the previous section. We construct graph MG as a pre-processing step. We then generalize MG to create a motion graph that supports interpolation. We call this

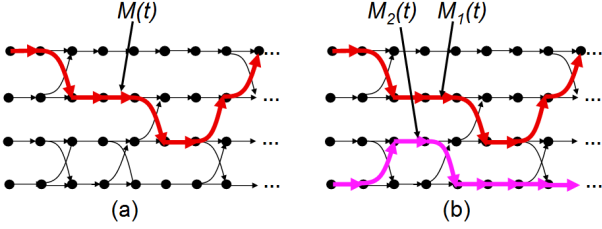


Figure 4: For this example, the database consists of four motions: two walks and two jumps. (a) Standard motion graphs find one path through the graph; (b) Interpolated motion graphs find two paths through the graph. The resulting motion is an interpolation of these two paths, $M_1(t)$ (red) and $M_2(t)$ (pink). $M_1(t)$ and $M_2(t)$ can transition independently between motions in the database.

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* would require less than 5MB).

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*.

5.3 Unrolling Into Environment

During the search, the graph is unrolled into the environment (by augmenting each state with the global position and orientation of the root). We call unrolled graph - *SG* (search graph). This step is required to search for motions that satisfy user-specified global position constraints and avoid obstacles. The process of unrolling is the same for both, standard motion graphs and interpolated motion graphs. In this section, we use motion graphs to describe unrolling.

This graph is built as the search works on it, thereby limiting the allocated memory to only the states that are actually computed. Each state in the graph *SG* is de-

fined by a pose in the motion graph *MG* and a global position and orientation of the root of the character in the environment. The global position is required in order to satisfy position constraints (such as getting to the goal). Depending on the problem, other variables could be added to each state (elapsed time, for example). Unfortunately, the addition of the position information increases the number of possible states significantly. For example, suppose the position of a character is defined by a 2D position in the plane (x and z) and an orientation about vertical axis (θ). Suppose also that each of these variables can take 1000 distinct values. If the motion graph *MG* has 10^4 states then the search graph *SG* will have $10^4 * 10^9$ states. This exponential expansion makes search in this graph difficult if not impossible.

During the search, in addition to augmenting each state with root position and orientation, we also augment each state with a constraint counter. The counter is used to ensure that all constraints are satisfied. 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.

6 Importance of Optimality

The search can be global but with no guarantees on optimality of the solution. In many cases, however, it is important to find a solution as close to an optimal as possible. Globally near-optimal solutions avoid the dithering and inefficient patterns of motion that greedy or locally optimal solutions often have [7].

Figure 5 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. Figure 6 shows jumping example. Optimal solution uses a running jump rather than a standing broad jump as a human likely would.

Tables 2 shows how the cost of the solution changes as its optimality increases during our search. Top table is for “walk from start to goal” example. The first solution is very suboptimal—the character makes two really large steps to reach the goal position (Figure 5). 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 (Figure 5). Middle table is for “walk with jumps over three river” example. The first solution is suboptimal—the character makes inefficient two legged jumps to cross all three rivers (Figure 6). The second solution is more natural, the character now uses a one-legged jump to cross the rivers (Figure 6). In the optimal



Figure 5: Optimal and sub-optimal solutions for walking a given distance (left) and for picking up an object (right).

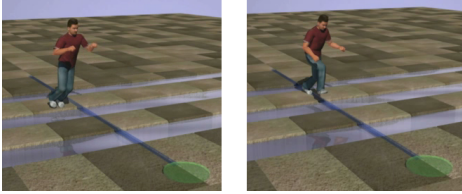


Figure 6: Optimal (right) and sub-optimal (left) solutions for jumping over river.

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 1: Importance Of Optimality

solution the character does not jump but steps over the last (the smallest) river. In the bottom table, the character starts on the small rectangle and needs to walk toward and pick a small object shown by a sphere in Figure 5. The first solution is very suboptimal, the character bends way too far to pick up a small object (Figure 5). The second solution is better but the character is reaching from the side which in the absence of constraints appears unnatural (Figure 5). The final solution is optimal and looks natural (Figure 5).

7 Optimal Search

We use A^* search [19], and in particular its anytime extension ARA^* [11], to find the paths through the motion graph. We briefly describe it in Section 7.1. In Section 7.2 we analyze complexity of our problem and show why it is hard. To make optimal search tractable we propose a lossless compression of the motion graph that significantly reduced the number of states (Section 7.3) and

a search heuristic that worked well for many examples of human motion (Section 7.4).

7.1 Search Method

We use A^* search [19], and in particular its anytime extension ARA^* [11], to find the paths through the motion graph and interpolation weights so that the interpolated path will satisfy the constraints and result in the optimal motion. The algorithm takes as input a graph 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 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 $s = s_{goal}$, $h(s) = 0$. 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 [20].

The anytime extension of A^* , ARA^* search [11], trades off the quality of the solution 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 its cost is bounded from above 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.

7.2 Complexity

The complexity of the A^* algorithm is $O(E + S \log S)$, 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. This graph 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 ($S = 10^{17}$ for this example 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 approximately 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 8, 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.

7.3 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 the optimal solution for any set of user-specified constraints 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 7.3.

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. Con-

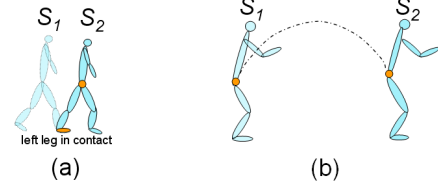


Figure 7: (a) For direct paths between a pair of contact change states S_1 and S_2 , the global position and orientation of the root of the character at state S_2 is uniquely determined by the contact position and orientation at state S_1 and the values of the joint angles at state S_2 . (b) The position 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 .

tacts are assumed to be not moving with respect to the ground plane or object in 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 [7] 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 [21, 22].

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 7). 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 8 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

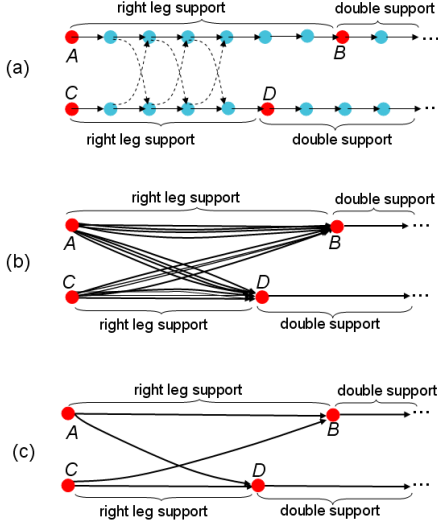


Figure 8: (a) States A , B , C and D (shown in red) are contact change states. If the character enters state A (and initiates a right leg support phase), it can exit only through state B or D . (b) A representation with only the contact change states shows that there are many paths between each state. (c) The graph after transitions are culled to include only optimal paths between contact states.

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.

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 independent of the particular 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 [9]). 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 dis-

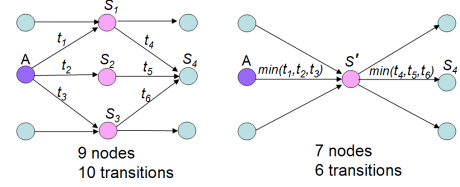


Figure 9: (Left) States S_1 , S_2 and S_3 are similar to each other. As a result, optimal transitions t_1 , t_2 and t_3 are also very similar and all end with the character at approximately the same position. (Right) We merge states S_1 , S_2 and S_3 into one state S' and keep only the lowest cost transition.

tance 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 [7]. With that approach no path would be found between states A and D in Figure 8(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, 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 similar. If we remove this redundancy, we can significantly reduce the size of the graph. This compression is performed on the graph that contains only contact change states, and the transitions are the optimal sequences of poses between contact change states.

For example, state A in Figure 9 has three successors, S_1 , S_2 and S_3 , that are similar to each other and all three transitions will end at approximately the same position in the environment when the graph is unrolled. We can cull the redundant states by merging states S_1 , S_2 and S_3 into one.

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 merging, we will have many redundant transitions. We keep only the lowest cost transition (Figure 9). We merge states in the order of their similarity. The simi-

larity threshold for merging can be substantially larger than that for establishing the initial transitions. A higher threshold for merging just removes flexibility from the graph whereas a higher threshold for transitions introduces perceptible errors. Because each transition in the compressed graph is a sequence of poses representing one contact phase of the motion, we can post-process each transition to remove foot-sliding as a preprocessing step.

Constructing Compressed IMG : Previous two section described how to compute a compressed standard motion graph. Compressed interpolated motion graph can be computed from it. After graph MG is compressed, graph IMG is constructed from it using the same process as described in Section 5.2. 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 (see [10] for more information). The interpolation weight w is constant throughout the transition. We use the interpolation scheme described in Safonova and Hodgins [18]. 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 formula in [18].

As was shown by Safonova and Hodgins [18], this interpolation scheme ensures that the majority of the created transitions in graph IMG are close to physically correct. We can also check these interpolated transitions for physical correctness using inverse dynamics at the time of construction of graph IMG .

7.4 Heuristic Function

We use an anytime version of the A^* search algorithm to find an optimal path in the unrolled graph, SG . 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 significantly reduce the amount of the search space that is explored. In this section, we present a method for computing such bounds. In Section 8 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 SG . We compute two heuristic functions: H_{pos} and H_{mg} . The first heuristic function, H_{pos} , ignores the dynamics of the motion of the character and estimates the cost of get-

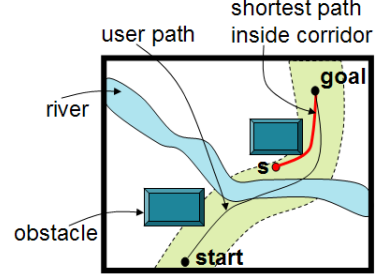


Figure 10: $H_{pos}(S, G)$ is the shortest path from the position of the character at state S to the goal. The shortest path is constrained to stay inside the corridor.

ting 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_{mg} , 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_{mg} and how to combine them. Same heuristic function can be used to search both standard motion graphs, SMG , and interpolated motion graphs, ISG .

Heuristic 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 10). 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 the paths with only a few milliseconds of computation. Because our cost function minimizes weighted average of energy and smoothness terms, we need to multiply the shortest distance (in meters) by an estimate of the minimum value in objective function required to traverse one meter. We compute the minimum value in objective function from the motion graph data. Because the computation of the heuristic $H_{pos}(S, G)$ depends on a given user sketch, it must be computed at runtime.

We use a coarse discretization to compute the heuristic function, but a much finer discretization when computing the unrolled graph SG . For that computation, the root position of the character was discretized into a 0.05 by 0.05 meter grid to avoid discontinuities in the final motion.

Heuristic based on motion graph state (H_{mg}): $H_{pos}(S, G)$ 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 an-

other. But if constraints are present then $H_{pos}(S, G)$ will underestimate the cost to the goal for two reasons: (1) user or environmental constraints usually require much more effort than the minimum torque estimate assumed by H_{pos} ; (2) the motion graph restricts what the character can do from a particular state, perhaps making a state that satisfies the constraint hard to reach. For example, if the character needs to jump over an obstacle and it is difficult to reach a jumping motion from state S , then the cost-to-goal at state S should be high. H_{pos} will grossly underestimate this cost and, consequently, A^* search will needlessly explore this part of the space. The second heuristic function, H_{mg} , addresses this problem by taking into account the capabilities of the character that are encoded in the motion graph. It estimates the extra cost (the cost not accounted by H_{pos}) of satisfying each type of constraint for each state in the motion graph.

In our implementation, we support five types of constraints: picking, jumping, stepping onto an obstacle (a beam for example), ducking, and sitting. The method should generalize to other types of constraints such as kicking, stepping over obstacles, or standing on one leg. For each type of a constraint supported by our system, $H_{mg}(S, C)$ is computed as the minimum cost of getting to any pose in the motion graph that satisfies the constraint C from the state S .

The computation of the H_{mg} heuristic does not depend on the particular constraint specified by the user and therefore can be precomputed. The computation of the H_{mg} heuristic is automatic because we have contact information for all motions in the motion capture database. We use the constraint of picking up an object to explain the computation of H_{mg} . The same method is used for all constraints supported by our system.

Both $H_{pos}(S, G)$ and $H_{mg}(S, C)$ account for the motion of the character in the plane. Therefore, the summation may overestimate the actual cost to the goal and violate the admissability requirement for the heuristic function. To resolve this problem, when computing the $H_{mg}(S, C)$ term, the cost of each transition in the motion graph is reduced by the minimum torque required to traverse the planar distance covered by the transition ($H_{pos} \text{ for that transition}$). We denote this heuristic by $\tilde{H}_{mg}(S, C)$.

When the constraint, C , is to pick up an object, $H_{mg}(S, C)$ represents the minimal cost of a path in the motion graph from state S to any state that represents the picking up of an object. Each “picking” pose can be defined by two parameters: *height* and *reach* (Figure 11(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

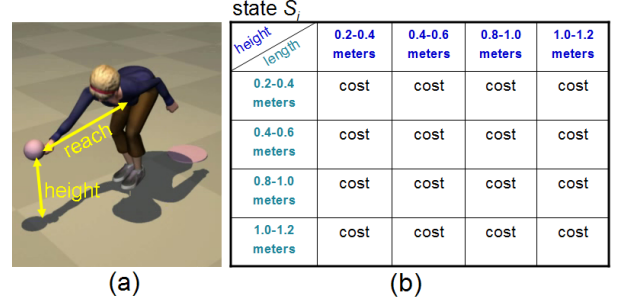


Figure 11: (a) We identify states where objects are picked up in the motion graph. Each such pose is parameterized by two parameters: *height* and *reach*. (b) For each state in the motion graph we precompute a table with the minimal cost of getting to a “picking” state with the specified height and reach parameters.

state in the graph MG that represents picking up an object and compute *height* and *reach* values for that state. For graph IMG , the process is similar. We identify each state $p = (I_1, I_2, w)$ that represents picking up an object. 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 the graph MG (process is the same for IMG), we then compute a table (Figure 11(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 that represents picking with *height* and *reach* values in this range. For each entry in the table and each state in MG , we search graph MG to compute the cost for that entry. The computation is really fast for graph MG . For graph IMG , for the database of 6-7 minutes of motion the precomputation of the H_{mg} heuristic for all constraints took less than an hour.

Combining the two heuristics: We combine $H_{pos}(S, G)$ and $H_{mg}(S, C)$ into a single heuristic function by summing them together. If at state S there are still n constraints remaining to be satisfied, we fetch the H_{mg} 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 :

$$H(S) = H_{pos}(S, G) + \sum_{i=1 \dots n} H_{mg}(S, C_i) \quad (2)$$

8 Experimental Results

To illustrate the effectiveness of our approach, we generated a variety of examples for both standard motion

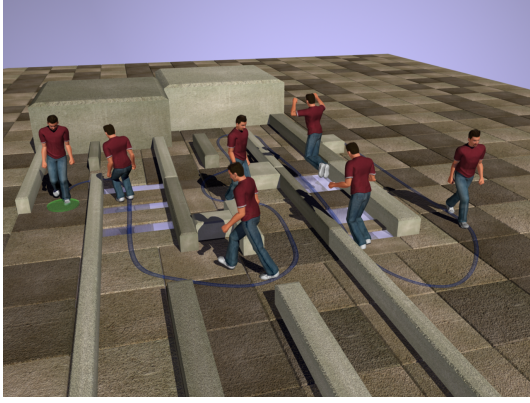


Figure 12: A forty-five meter motion with jumps and walks

graphs (*MG*) and interpolated motion graphs (*IMG*). 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.

8.1 Search Effectiveness for standard motion graphs (MG)

We have generated a variety of different examples including various walks such as straight walks and walks with slow and sharp turns, stepping over stones and jumps of different lengths and of different types (one-legged jumps and two-legged jumps). The figure 12 shows a screenshot of one of the motions generated by our algorithm.

For all the experiments we used a motion database containing approximately 12,000 frames of human motion captured at 30 frames per second. The motions included various walks, jumps, and picking up objects motions. The motion database was used to generate a single motion graph for all the experiments. The generated motion graph was relatively densely connected with the number of states being the same as the number of frames and the number of edges equal to about 250,000. While the density of the transitions in general makes the task of the search harder, it has a significant benefit in that we can generate motions that are quite different from the motions in the motion database. This freedom allows motions to follow the user specified path and meet the constraints well. In addition, because the generated solutions minimized the sum of squared accelerations (an approximation to energy), the motions generally avoided various artifacts such as dithering back and forth motions. All the generated motions were post-

Search Time (secs)	Solution Cost	Optimality Bound (ϵ)
0.016	10658960	10.0
0.031	10658960	2.1
0.032	8226729	2.0
0.033	8226729	1.4
0.250	6268591	1.3
0.844	6268591	1.0

Table 2: The cost of a solution and the bound on its sub-optimality as a function of search time. (The numbers are only given for the time points when the solution cost changed.)

processed to remove feet sliding, an effect usually seen in motions that come straight from a motion graph. (The post-processing used a very simple local optimizer.)

The lengths of the motions generated by our approach varied from 8 to 60 seconds. The longest one was a 45 meter walk through a maze that also involved a number of examples of jumping over water. In all the examples the search quickly generates a first solution (for sub-optimality bound ϵ set to 10): usually within 2 to 3 seconds. In case of the long walk through the maze though the first solution was generated within 30 seconds. The search then improves the solution within the remaining time allocated to it. For each example the search was given 120 seconds to find the best solution it can. The table 2 shows how the cost of the found solution improves during the search for one of the simpler experiments. ϵ set to 1 corresponds to a provably optimal solution. Usually, the search converged to somewhere in between $\epsilon = 1.1$ (at most, 10% sub-optimality) and $\epsilon = 1.5$ within the allocated time.

8.2 Search Effectiveness for interpolated motion graphs (IMG)

Figure 1 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 13 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.

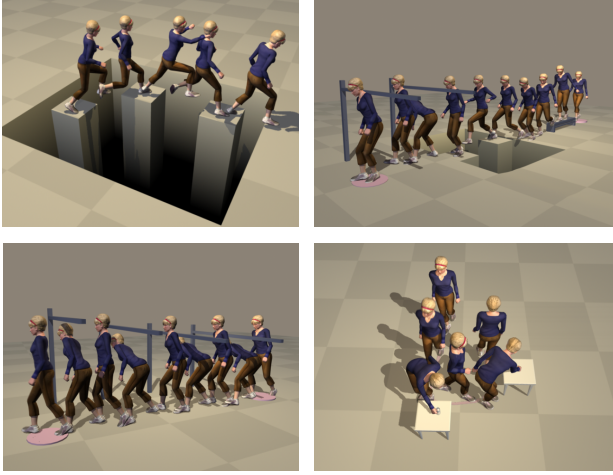


Figure 13: Synthesized motions

	Before merging	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 3: Compression for three motion graphs. The first graph is computed from walking, jumping, ducking, sitting and walking along the beam motions. The second graph is computed from walking and picking motions and the third one is computed from just walking motions.

8.3 The benefit of motion graph compression

In this experiment, we evaluate the effect of motion graph compression. Table 3 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 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.

8.4 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) the H_{pos} component of our objective function; (3) the H_{mg} component of our objective function; (4) the combined heuristic function with both H_{pos} and H_{mg} 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.

9 Discussion

Motion graphs and their variations have proven to be a powerful technique to solve for a desired motion when only rough sketch is given. In this paper, we have demonstrated that it is possible to search a standard motion graph and interpolated motion graph using a globally optimal search algorithm, A^* . Two contributions made this possible: a lossless compression of the motion graph that significantly reduced the number of states and a search heuristic that worked well for many examples of human motion. We demonstrated that the global search was effective by creating long example motions and showing that the optimal and near-optimal solutions avoided the dithering and inefficient patterns of motion seen in many other motion graph implementations.

Because the method computes a compressed motion graph that contains only optimal paths, variations that may have existed in the original data may be lost. Variations are always “sub-optimal” and therefore 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 is 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. Our experiments show that to obtain good results many states must be able to quickly connect to the constraint states and vice versa.

Global optimization has two significant effects on the motion of the character. First, it should iteratively find the “correct” strategy for the character to use to navigate an environment. For example, is a two-legged jump or a one-legged jump more efficient for an obstacle of a given size? Table 2 illustrates these discrete changes. The second feature of the global optimization should be to fine tune the motion, choosing a series of walking steps with little velocity change, for example. This second feature is not as apparent in the animated mo-

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 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.

tion but is still visible in the decrease in energy as the optimizer iterates.

Although we did a few informal experiments to see if a subject used the same strategies as the animated character for a given terrain, we did not do a definitive assessment with a large naive subject pool. Such a study would be easy to run (using coding of the behavior selected for each part of the obstacle course as the metric). We expect that the sequence of behaviors from the human subjects would be similar to those of the animated character for many but not all examples. They might differ because the motion graph did not include the right behaviors (a long one-legged jump, for example) so another less efficient behavior is selected (a long two-legged jump, for example). Alternatively differences might arise because people do not always optimize efficiency but instead optimize for style, comfort, safety or other factors.

References

- [1] A. Witkin and M. Kass, “Spacetime constraints,” *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, no. 4, pp. 159–168, 1988.
- [2] A. C. Fang and N. S. Pollard, “Efficient synthesis of physically valid human motion,” *ACM Trans. on Graphics*, vol. 22, no. 3, pp. 417–426, 2003.
- [3] A. Safonova, J. K. Hodgins, and N. S. Pollard, “Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces,” *ACM Trans. on Graphics*, vol. 23, no. 3, pp. 514–521, 2004.
- [4] A. Sulejmanpašić and J. Popović, “Adaptation of performed ballistic motion,” *ACM Trans. on Graphics*, vol. 24, no. 1, pp. 165–179, 2005.
- [5] Y. Li, T. Wang, and H.-Y. Shum, “Motion texture: a two-level statistical model for character motion synthesis,” *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 465–472, 2002.
- [6] L. Kovar, M. Gleicher, and F. Pighin, “Motion graphs,” *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 473–482, 2002.
- [7] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, “Interactive control of avatars animated with human motion data,” *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 491–500, 2002.
- [8] O. Arikan and D. A. Forsyth, “Interactive motion generation from examples,” *ACM Trans. on Graphics*, vol. 21, no. 3, pp. 483–490, 2002.
- [9] O. Arikan, D. A. Forsyth, and J. F. O’Brien, “Motion synthesis from annotations,” *ACM Trans. on Graphics*, vol. 22, no. 3, 2003.
- [10] A. Safonova and J. K. Hodgins, “Construction and optimal search of interpolated motion graphs,” in *ACM Trans. Graph.*, p. 106, 2007.
- [11] 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.
- [12] K. Pullen and C. Bregler, “Motion capture assisted animation: texturing and synthesis,” *ACM Trans. on Graphics*, vol. 22, no. 2, pp. 501–508, 2002.
- [13] M. G. Choi, J. Lee, and S. Y. Shin, “Planning biped locomotion using motion capture data and probabilistic roadmaps,” *ACM Trans. on Graphics*, vol. 22, no. 2, pp. 182–203, 2003.
- [14] M. Sung, L. Kovar, and M. Gleicher, “Fast and accurate goal-directed motion synthesis for crowds,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 291–300, July 2005.
- [15] M. Lau and J. J. Kuffner, “Behavior planning for character animation,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 271–280, 2005.
- [16] M. Lau and J. Kuffner, “Precomputed search trees: Planning for interactive goal-driven animation,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 299–308, Sept. 2006.
- [17] J. Lee and K. H. Lee, “Precomputing avatar behavior from human motion data,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 79–87, 2004.
- [18] A. Safonova and J. K. Hodgins, “Analyzing the physical correctness of interpolated human motion,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 171–180, 2005.
- [19] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
- [20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 2003.
- [21] L. Ikemoto, O. Arikan, and D. Forsyth, “Knowing when to put your foot down,” in *ACM Symposium on Interactive 3D Graphics*, pp. 49–53, 2006.
- [22] B. L. Callenec and R. Boulic, “Robust kinematic constraint detection for motion data,” in *ACM SIGGRAPH/Eurographics Symp. on Comp. Animation*, pp. 281–290, 2006.