Solving computer animation problems with numeric optimization

Victor B. Zordan

College of Computing and Graphics, Visualization, and Usability Center
Georgia Institute of Technology
Atlanta, GA 30332-0280
victor@cc.gatech.edu

Abstract

Optimization techniques have been used in computer animation to search for system parameters and control inputs in a variety of animated objects and characters. Researchers rely on numeric optimization to solve problems including behavior learning and morphology generation for characters as well as automatic tuning for weights and parameters in various models. However, even with the diverse group of published examples, the selection of an optimization technique for new problems can be difficult for inexperienced animators. The choice of appropriate methods and their proper implementation requires an understanding of the types of methods and their respective advantages and limitations. Toward this end, I describe a general approach for formulating an optimization problem to help organize the information pertinent to the selection process and provide a common vocabulary for discussing the issues related to this type of problem-solving. I provide a straightforward classification of optimization methods and discuss characteristics and trade-offs related to the algorithms. Then, I describe specific uses of the methods with results from recent works in computer animation. I detail solutions for two common optimization problems namely, inverse kinematics and control gain tuning, and make general recommendations about solving optimization problems in computer animation in closing.

Keywords

Computer animation, optimization, physical modeling.

I. Introduction

Numeric optimization methods are powerful search tools that have been employed to solve a variety of problems in computer animation. But, even with the rich library of examples described by the animation community, selection of an optimization method for a new problem can be difficult without relevant background knowledge. This report presents optimization techniques in a simple-to-complex ordering, followed by how the techniques have been used in previous approaches and how they may be used to solve new problems in computer animation.

Numeric optimization, used extensively in disciplines such as engineering and robotics, has been introduced to computer animation in a case-by-case manner. I structure these individual examples according to a classification, drawn from engineering design and machine learning, to provide a reference for animation researchers. While optimization methods fit into two main categories based on the types of problems they are used to solve, namely parameter-tuning problems and control approaches, this report focuses primarily on parameter-tuning problems and will mention control techniques only for comparison. I organize this classification according to the type of information each technique requires from the system of interest.

The report starts by describing how to generate a well-formed problem statement, called the problem formulation. Through the formulation, the animator specifies the parameters to be modified, the constraints that must be satisfied, and the goals or objectives to be met during the parameter search routine. By organizing this information, characteristics about the problem can be identified and the search method to be used is more easily selected. Numeric techniques for solving a formulation are presented next, in a logically ordered classification. Algorithms which require no knowledge from the system of interest are described, including searches that are random and patterned according to heuristics, followed by gradient-based methods. The report then outlines computer animation literature that has used the described techniques and how to use the techniques in new problems. Previous works are summarized including the type of problem being solved, the parameters being searched for, and the objectives used for identifying good solutions. I include two samples of common problems in animation and conclude with solutions to these problems as well as comments about selecting optimization techniques for other problems.

II. Formulating the optimization problem

The problem statement, or formulation, of an optimization problem plays a key role in the selection of a suitable technique for solving the problem. A straightforward four-step formulation can, in general, be posed in the following form:

> **Given** a *model* for the system of interest,
> **Find** the value of the selected *parameters* that
> **Satisfy** the hard and soft *constraints* and
> **Minimize** the given *objective function*.

From characteristics discerned from this formulation, the animator matches the problem to an appropriate optimization routine.

Problem formulation begins with a model of the object of interest. This model in most engineering settings is a representation of the actual system of interest, for example a simulation of a turbine to model its physical counterpart. However, in computer animation the model usually is the exact system of interest and optimization may be applied directly to this system. At times, even in computer animation, a simpler model may be used for a particular search as long as the simple system responds to the optimized parameters in the same way that the more complicated system does. Some examples of common models in computer animation include kinematic models for animated characters and physical models for simulated liquids, flexible and rigid body systems.

Given a model, the next step in formulating the optimization problem is to select the input or control parameters to be modified by the search. Here, the assumption is that some combination of the selected parameters will lead to a desirable solution. For example in a kinematic model, the joint angles may be modified to produce a posture with particular features. Input parameters may be discrete or continuous. However, when choosing an optimization method, discrete parameters will require special consideration because they cannot provide gradient or slope information directly. Also, the number of parameters has a strong effect on the choice of methods and success of an optimization process. As a rule, the smallest set of input parameters that produces a reasonable solution is the correct choice.

Next, hard and soft constraints may be added to the formulation. The optimization routine must satisfy hard constraints for a solution to be valid while soft constraints may be violated but the objective function should penalize for the degree of the violation. Hard constraints may be used, for example, on the maximum fuel consumption or top strength of a character's muscles while soft constraints may be used in cases where violations may be tolerated like the max speed of a moving object or the flexion of a physically modeled spring. At times, the choice of hard verses soft constraints is left to the researcher's discretion. For example, joint limits in a kinematic model may be taken as hard or soft constraints depending on the needs of the application. Constraints limit the types of optimization methods that may be used in solving problems because some methods cannot handle them. In some cases, constraints may be transformed into input parameters using Lagrange multipliers [18]. In this fashion, an initially constrained problem may be solved using unconstrained techniques at the cost of making the problem more complex.

The final step in formulating the optimization problem is selecting the calculable goal or goals used to score a potential solution. These goals are combined into a formula called the objective function and the score is called the performance index. The numeric algorithm uses the objective function to assess the performance index of a given solution. Therefore, this function should be chosen carefully. When the objective function is minimized, the corresponding solution is expected to be better than the other solutions evaluated in terms of the researcher's definition of goodness. A naive choice in objective functions can lead to undesirable but numerically *better* solutions. For example, in the problem of picking control inputs for a jumping motion, a naive objective function may be to minimize energy. However, the best solution in this case could end up with no jump at all because this solution expends the least amount of energy.

Objective functions may include single or multiple goals. While some optimization techniques handle multiple objective functions, it is common to combine the goals into a single objective using an Archimedian weighting scheme. In this weighting scheme, the single objective function is comprised of a linear combination of the individual goals. In the jumping example, the objective function may be the weighted sum of the minimum energy as well as maximum distance covered. Proper selection of the weight constants for each goal is required to handle differences in units and assign the proper importance to each particular goal.

To help clarify the method of formulating the optimization problem, I detail formulations for two animation problems, specifically inverse kinematics and control gain tuning in Section V. Given the problem formulation, the selection of an optimization method can be made by assessing characteristics about the model, the choice of input parameters, the smoothness of the solution space, and the contributing members of the objective function. Considerations that lead to a particular optimization process are described for individual methods in the next section. With a concise, well-developed problem formulation and a suitable technique, implementation is often straightforward.

III. Classification of optimization methods

Numeric optimization has been used in a wide variety of fields including economics, mathematics, and engineering as well as various areas within computer science such as game theory, robotics, and computer graphics. The optimization methods presented here are drawn from standard methods used in applied engineering design and control. Basic descriptions of the techniques are included here, but the general engineering text by Reklaitis, Ravindran, and Ragsdell [18] and the control text by Dyer and McReynolds [7] provide more thorough discussion and analysis of many of the techniques. I describe parameter search techniques and conclude with a brief synopsis on techniques applied to control.

In this report, I classify numeric optimization methods by the information each requires of the system model. A numeric optimization formulation assumes that the system model may be evaluated using the objective function for a given set of input parameters. Depending on the information available from the model, optimization methods can solve the search problem with differing degrees of success. I discuss three basic classes of methods: direct search, gradient-based methods, and second-order methods. In direct search, the model is assumed to be a black box, receiving a stimulus and providing a response. Gradient-based methods assume that useful gradient information exists and that it may be determined analytically or numerically from the model. Similarly, second-order methods require that the model contains useful second-order gradient information.

A. Optimization methods employing direct search

The most straightforward direct search method is to perform an exhaustive search of all of the possible scenarios and select the solution that yields the best performance. This form of direct search is guaranteed to find the best global solution according to the problem statement. While in simple cases an exhaustive search can be a satisfactory technique, its utility quickly degrades as the optimization problem becomes more complex. As the number of input variables increase, the searches required increase by the exponent of that number. With even a small number of input variables (often less than ten), the search often becomes too expensive computationally or simply infeasible.

If the problem is too complex for the solution space to be searched exhaustively, an alternative approach that does not require gradient information is to search randomly. A number of useful algorithms follow this paradigm, some using information gathered from successive tests to make better selections during the process. The simplest stochastic search method is aptly called random exploration and can include either global or local exploration. Global exploration is performed by randomly generating feasible input samples, performing queries for each and selecting the input with the best performance index. Conversely, local exploration, also called random walk, starts at some initial input (determined randomly or specified by the user) and takes fixed-size steps in random directions. In this algorithm, a new step is selected at random and accepted if the corresponding performance index is better than that of the previous step.

Two algorithms based on the premise of random exploration have been used successfully in a variety of problems, namely simulated annealing and genetic algorithms. Simulated annealing performs a random walk and rejects or accepts steps according to a changing probability. The annealing algorithm, named after a heat-treating process used to grow large malleable grains in metal by slow-cooling, rejects steps with a lower performance index according to a slowly increasing probability. By initially accepting poor steps at a probability, the algorithm is able to avoid local minima in search of better, more global solutions. As the annealing schedule lowers the acceptable "temperature", poor steps are thrown out with a higher probability and the process converges on a solution. For a detailed description and sample implementation of simulated annealing, see Numeric Recipes [17]. Annealing is capable of finding improvements in discontinuous, sparse, or rough solution spaces when little is known about the underlying space. Unlike many other algorithms, annealing seamlessly mixes discrete and continuous search variables. However, the algorithm requires many queries which can be too slow for real-time or interactive applications often found in computer animation.

Genetic algorithms employ a modified version of local random exploration through the use of a biological analogy to evolution. In this paradigm, each set of discrete input variables is considered to be a unique code, similar to a set of genes. The algorithm generates a random population of inputs, iteratively tests the population, and combines sets that perform well. By combining genes from successful predecessors, the algorithm *evolves* better inputs during successive generations. This combining process can be straightforward, taking genes from each parent to generate a new set, or complex with mutations, crossovers, or combinations of both. In either case, the success of the algorithm depends on the assumption that the factors that make a good gene set will not be lost by changing some of the genes. Genetic algorithms, like simulated annealing, accept input variables that are discrete or continuous, and can handle jumps and discontinuities in the solution space.

Other than random or exhaustive exploration, one other direct search method incorporates heuristic hill climbing. This class of techniques attempts to make efficient steps toward a local solution through the use of knowledge collected from repetitive queries with the cost of computing the gradient avoided. In this case, the solution space is assumed to be continuous. This class includes algorithms for pattern searches and Powell's method. In pattern search, the resulting search path follows a particular *pattern* in each cycle. For example, in the pattern search presented by Hooke and Jeeves, the algorithm assesses the local area, takes steps in a single direction until improvement ceases, and then repeats the cycle [18]. The approach in Powell's method attempts to quickly find a good solution by successively sampling the local area and fitting it to a quadratic function. The algorithm computes the minimum of this quadratic and steps to the approximated minimum and the process repeats. The assumption in this approach is that if a minimum exists which is not on the boundary of the solution space, then the surface around the minimum must fit a quadratic. Narrowing in on the solution, the estimated quadratic can fit the local area more tightly with each iteration. A modified version of Powell's method, named conjugate directions, attempts to align the axes of the quadratic with the local geography to reach a solution even more quickly than simple Powell's.

B. Gradient and higher-order optimization methods

Unlike direct search, gradient-based methods use higher-order information to navigate the solution space. These algorithms require the gradient or derivative of the objective function with respect to the input variables. This gradient may be computed through exact calculation or approximation such as finite differences. In general, these techniques tend to take fewer iterations to find solutions than direct search because the algorithms make more intelligent steps. However, gradient methods use knowledge about the local area of the solution space and do not make global jumps. Therefore, solutions found are local minima and are not guaranteed to be the best globally.

A simple and popular technique that employs gradient information is called Cauchy's method or greedy search. Computing the gradient or slope at a given starting point, the algorithm iteratively finds the direction of steepest descent and moves in that direction until no further improvement is made. A non-zero gradient guarantees that there is improvement to be made and the method attempts to move closer to the local minimum, usually quite quickly initially. But near the minimum, as the gradient drops off, the algorithm approaches the precise extrema slowly.

By employing second-order information, techniques such as Newton's method can make more knowledgeable choices and avoid the difficulty of slow termination as observed in the greedy algorithm. Newton's method estimates the local solution space with a truncated Taylor expansion. This expansion requires a local value for the second-order gradient, or Hessian, of the solution space. Unlike greedy search, Newton's method allows steps with a negative gradient as the algorithm moves directly to the point in the solution estimated to have a zero gradient. This approach usually terminates more quickly than Cauchy's method. One downfall of Newton's method is that the Hessian is expensive to compute or undefined at times, especially in high-dimensional searches.

A popular set of approaches that attempt to provide the benefits of Newton's method using only first-order information include conjugate-gradient and quasi-Newton methods. Conjugate gradient differs from Cauchy's method in that the algorithm attempts to select directions for each step by considering previous directions as well as the gradient. By choosing step directions carefully, conjugate gradient moves directly toward the minima and avoids the repetitive sweeps near the minima seen in Cauchy's method. Several variations of quasi-Newton methods have been introduced. Each mimic Newton's second-order method by approximating the Hessian of the objective function but they differ in the way in which the approximation is computed. These quasi-Newton algorithms are among the most popular gradient-based methods applied in engineering and computer animation applications.

C. Optimization methods for control

In control design, the optimal controller for a particular system is defined to provide the best performance for the system according to the objective function. While theoretical optimal solutions exist for linear systems, many systems, particularly non-linear ones, require numeric techniques. Two popular methods for solving control optimization are dynamic programming and quadratic programming. In dynamic programming, the control problem is formulated as a sequence of stages with a single optimization problem at each stage. The dynamic programming algorithm treats each stage independently and *floods* forward one stage at a time. A more sophisticated variation includes a secondary sweep method that floods forward initially and corrects using a backward pass through the stages. A second approach for solving the multiple sequential optimization problem for control is quadratic programming. Unlike dynamic programming, quadratic programming considers the entire trajectory during an optimization process.

IV. Review of previous animation results that use optimization

In this section, I summarize results from previous works in computer graphics where animation researchers have used numeric optimization. The descriptions of these efforts are put into categories following the classification detailed in the last section.

A. Animation techniques using direct search

Researchers in computer graphics have explored direct search in the form of stochastic parameter optimization to animate several models. Stochastic techniques are often chosen instead of gradient methods in situations where global solutions are needed or where gradient information is unavailable or too expensive to compute. Simulated annealing and genetic algorithms have been used to solve a wide spectrum of problems in computer animation. Here, I describe several implementations of these two algorithms.

Simulated annealing is a popular technique for solving computer animation problems because it is easy to implement and quite flexible. Grzeszczuk and Terzopoulos propose learning control parameters for physical models using simulated annealing [10]. In this work, simulated creatures, such as fish and snakes, use muscles actuated by spring networks to perform low-level behaviors such as forward locomotion and turning, as well as high-level behaviors such as path following. At the low-level, they use simulated annealing to solve for linearly interpolated control inputs in a time-based controller and for parameters of sinusoidal basis functions in frequency-based actuators. At the high level, they combine the low-level controllers using annealing to select order and duration. The latter problem mixes discrete and continuous parameters. Their objective functions for both problems minimize measures for distance, speed, and energy efficiency.

Hodgins and Pollard use simulated annealing to adjust controller parameters as well [12]. Their technique uses direct scaling to modify an existing controller from one simulated humanlike character to another followed by annealing to tune a small number of parameters for the scaled controller. In this fashion, they are able to modify running and biking controllers for characters with different dynamic and kinematic properties. During the tuning phase, they search for high-level parameters such as the pitch of the body and ankle thrust in running. Their objective function includes features such as head acceleration and stride regularity. This search space is not smooth, making local and gradient techniques less useful. Simulated annealing, on the other hand, is able to move through this space and find reasonable solutions.

A third use of simulated annealing is described by van de Panne and Fiume [21]. This work uses random exploration followed by simulated annealing in a fine-tuning phase to generate sensor-actuator networks for physically based models. They represent controllers as a network of weighted connections between the actuators of a creature and simple binary sensors for touch, orientation, length, and sight. Their algorithm initially generates network weights randomly and evaluates the corresponding motion according to straightforward metrics such as distance traveled and target errors. Successful creatures go through a simulated annealing phase to make small adjustments from the initial weights to improve performance. By keeping the step size and the number of iterations for the simulated annealing small, they force the algorithm to act as a local search. Their method results in successful locomotion and target-following for a variety of two-dimensional figures.

Because of the biological analogy, genetic algorithms are particularly appealing for generating animated creatures. Ngo and Marks introduced genetic algorithms for generating motion to the animation community [15]. Their approach automatically generates stimulus-response controllers for characters using genetic algorithms. In their results, they include motions such as leaping, skipping, and walking for planar stick figures also using simple evaluation metrics such as distance traveled. Sims used genetic algorithms for generating animation as well [20]. His work is distinct from that of Ngo and Marks in that his efforts were directed at evolving 3D creature morphology in addition to control. In order to evolve morphologies, virtual creatures were described through directed graphs and a number of geometric parameters. Simulated motions were rewarded for fitness in tasks such as swimming, jumping, locomoting, and target-following. Although the results in this effort are compelling, the lack of user control over the evolution process makes generation of a character with desired features difficult.

B. Animation techniques using gradient-based search methods

Gradient-based techniques appear repeatedly throughout the computer animation literature. I describe a variety of different uses to expose the strengths of gradient techniques, ordered here from simple to complex. van de Panne and Lamouret use a straightforward greedy descent algorithm to determine desired values of joints for a locomotion controller [21]. Their approach uses multiple stages of optimization so that each search relies on the results of the previous. By starting with an easy task and moving to a harder one, they can use a simple search at each stage because the search space is smooth. In order to generate a walking controller that balances, they initially provide a helping force to balance the model. They reduce this force through successive search stages. They report results for a simple character that locomotes while balancing without a helping force but they were unable to find solutions for more complicated, humanlike characters that balance. van de Panne and Fiume describe another example of using a straightforward descent algorithm in their sensor-actuator work and compare the algorithm to simulated annealing [21]. Bruderlin and Calvert use simple gradient searches as well to generate a hybrid kinematic and dynamic walking model [5].

Implementations of more sophisticated gradient searches are described by some researchers. For example, Newton's second-order method is employed on a locomotion controller described by Pai [16]. In this work, a walking character is produced by specifying a small number of loosely defined constraints. Pai uses automatic differentiation which allows fast computation of gradient information, making Newton's method feasible. In general, however, due to fast and reasonable approximations for computing the Hessian, Newton's method has not been used often in computer animation.

Grzeszczuk, Terzopoulos, and Hinton propose approximating physical models in order to speed up computation both at run-time and when constructing controllers [11]. They suggest an approach for training neural networks that employs both greedy and conjugate-gradient algorithms to tune parameters. With conjugate directions, they report improved performance over simple gradient descent. In addition, they note that the neural networks that emulate their physical systems allow partial derivatives to be computed reliably which aids them in searching for controller parameters.

Two examples of a quasi-Newton method appear in efforts by Rose and his colleagues [19]. Their contribution focuses on the semi-automatic generation of transitions between segments of motion capture data. They use the Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton method both in preprocessing human motion data and in generating transitions between sequences of data. Inverse kinematics is employed in order to fix contact between the feet and the ground using the algorithm. Because the motion is initialized from recorded human data, a solution close to the starting position is desirable and afforded by the local gradient search. When creating transitions, the BFGS routine is also employed for a fast solution to their inverse dynamics formulation. They report that BFGS allows the solution to be found within a few iterations, considerably faster than a randomized search.

C. Animation techniques using optimal control and quadratic programming

Only a few works in computer animation report approaches using optimal control and the methods appearing have applied dynamic programming to very simple animation problems. One of the first works to propose optimal control for animated models comes from Brotman and Netravali and applies control to a simple point mass and a single body vehicle [4]. The work of van de Panne, Fiume, and Vranesic includes slightly more sophisticated models such as a truck with a movable cab, and the classic cart and pole problem from robotics [22].

Quadratic programming has been used more often for control of physical models in animation. A trend in this research is labeled as spacetime constraints after the seminal work of Witkin and Kass [23]. In computer graphics, spacetime constraints has been used to solve a variety of animation problems. Witkin and Kass use quadratic programming to solve for mechanically efficient motion in the animation of a physically simulated jumping Luxo lamp. Constraints for their system include user-specified positions, velocities at key-times as well as limited muscle actuation for the lamp model. The problem is formulated to minimize the power consumed by the "muscles" while avoiding violations to the specified animation constraints. Results from this work show examples of jumps with changes in parameters of the physical model and user-specified constraints.

Most physical models discussed for computer animation are initialized by a user and run to completion without input from the user. This lack of user control is addressed in Cohen's work on interactive systems [6]. Cohen presents an optimization system in which a user specifies constraints such as positions, joint configurations, and time-varying trajectories to control a dynamic system. As in other work in spacetime constraints, quadratic programming is used to compute the desired animation. Examples include two and three link systems such as a one-arm

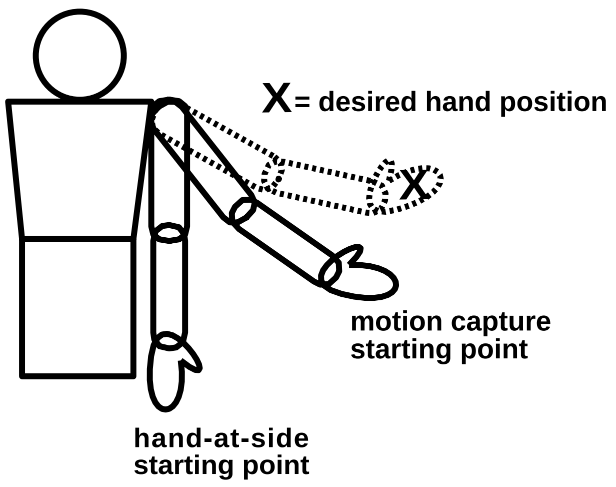

Fig. 1. **Inverse kinematics for hand position.** Inverse kinematics modifies the joint angles in the arm to satisfy the desired hand position, determined by the task being performed. For comparison, two starting configurations are tested, hand-at-side as a naive starting point and the more knowledgeable starting point from motion capture data.

basketball player that throws and catches. Another approach focusing on interactive spacetime constraints is described by Gleicher [8, 9]. His goal is to provide a user with an interactive system for modifying motion capture data. Gleicher trades physical constraints for a fast solver and interactive editing. In this case, the objective function minimizes the overall difference between the generated motion and the original captured data. Quadratic programming is used to satisfy desired changes such as hand and foot holds, as well as kinematic parameters such as limb lengths. Examples in this work include interactive motion editing for characters that change step location in walking cycles and adapting to changes in scale while swing dancing.

V. Formulating and solving two common animation problems

In this section, I formulate and solve two animation-related problems, inverse kinematics for a three link arm and gain parameter tuning for a balance control in a physical humanlike model. I present solutions to these problems using multiple optimization algorithms for each problem and discuss the benefits and differences of each method.

A. Inverse kinematics for hand positioning

An inverse kinematics solver selects joint angles for a chain of rigid links that satisfy a given Cartesian position for the end effector of the chain. This is a common problem in computer animation, for instance, placing a character's hand so that it may make contact with an object in its environment. Inverse kinematics often does not have closed-form solutions in high degree-of-freedom systems because of redundancy and singularities. Kinematic redundancy arises when more degrees of freedom exist than are needed to find a single solution. In this case, multiple solutions satisfy the under-constrained problem. Elbow positioning in arm postures is one example where the hand can be placed in a particular location but the elbow is still free to rotate in space. Singularities are special configurations where the link's movement is impeded in a particular direction, such as when a joint folds back on itself. In this scenario, the outer (or outboard) limb may not move forward or back without first moving out of the overlapped configuration. Baillieul and Martin provide a good discussion of these and other related issues [2].

One application of inverse kinematics is the modification and clean up of human motion data. Figure 1 shows a schematic of this problem for hand positioning. In magnetic motion capture data, joint angles and body positions are both available. While the angles alone may be used to create a posture, the position of the end effector would depend on the limb lengths of the animated character. Sometimes, the position of the hand marker is more important that the joint angles. An inverse kinematics solver is used to modify the joint angles to satisfy desired hand positions. When motion capture data for the joints is used as an initial guess, a local solution for the proper hand position is reasonable, assuming that staying close to the original posture is a good notion.

The formulation for the optimization problem requires a model of the forward kinematics for the character, including limb length and joint center positions. No explicit joint limits are imposed in this case. In general specifying limits for joint angles helps maintain solutions that are more like human postures. However, assuming the character is not very different from a human in its proportions and that the the initial conditions are specified from human motion, neighboring or close-by joint postures are usually satisfactory without explicit joint limits.

In inverse kinematics, optimizing multiple goals helps handle redundancy and singularities. For example, by specifying a target joint configuration, all of the solutions that satisfy the end effector goal may be compared to this target and a single best solution may be resolved. The problem formulated here closely matches the approach discussed by Bodenheimer and colleagues [3]. To solve the inverse kinematics problem formulated, three terms are included in the evaluation function: hand position error; hand orientation error; and elbow position error. Hand position and orientation allow for the desired placement of the end effector that may be specified by the user, or by the original position and orientation of the data recorded from the motion capture. The elbow position error is used to resolve the redundancy problem and maintain a *natural* posture. An Archimedian weighting scheme is used to combine these three goals.

This inverse kinematics optimization problem formulation can be summarized as follows:

Given

- Kinematic model
- Motion capture data
  - joint angles (used for initial guess)
  - body positions (used for pos errors)
- Target hand position (desired goal)
  - taken from motion data or user-specified

Find

- System Variables θ_i i = 1 to 9
  - Shoulder XYZ Euler angles
  - Elbow XYZ Euler angles
  - Wrist XYZ Euler angles

Satisfy

- Kinematic Constraints
  - Imposed by forward kinematic model

Minimize

- Single Objective Archimedian Scheme
  - $\omega_1 + \omega_2 + \omega_3$
  - ω_1 = hand position error
  - ω_2 = hand orientation error
  - ω_3 = elbow position error

To solve the inverse kinematics problem, I implement two local search methods, one gradient and one direct search. A local search is appropriate in this case because the motion capture data provides a reasonable initial guess. While the gradient method will likely find a better solution, a time critical application may dictate a faster solution because the calculation of the gradient may be too expensive. Specifically, I compare the performance of BFGS, a quasi-Newton method, and the Hooke and Jeeves pattern search algorithm to solve this problem.

The implementation of the BFGS algorithm iss from Numerical Recipes [17]. The algorithm requires a gradient for the function being minimized and I employ a finite difference approximation using the Riddler algorithm also from Numerical Recipes. Riddler includes testing the gradient multiple times to be sure the approximation is reasonable and making the computation of the gradient more reliable. Results for the BFGS algorithm are included in Figure 2. To show that the motion capture provides a reasonable initial guess, a hand-at-side "zero" position is used as a naive initial guess for comparison. Figure 3 shows the algorithm's performance with two different starting configurations. When the initial guess is taken from the motion capture data, a shorter path through the solution space results and the algorithm converges more quickly.

To compare the overall performance of the BFGS algorithm, I implemented Hooke and Jeeves as a direct search method to solve the same inverse kinematics problem. I chose this pattern search algorithm as a basis for comparison because it did not require a gradient calculation, which was slow and not always reliable in the BFGS solution. The two algorithms converged to about the same end configuration when the initial joint angle configuration was taken from the motion capture data. The error for a trial is compared in Figure 4. This figure shows the Hooke and Jeeves search wandering slowly before terminating, indicating a larger error tolerance would lead to faster termination. I repeated the experiment for the naive initial joint angle posture and the results comparing the two algorithms are shown in Figure 5. This comparison revealed that Hooke and Jeeves converged faster than BFGS, most likely because the gradient information from this posture was less useful. Also, in terms of compute time, because the gradient was not calculated, the Hooke and Jeeves approach took about a quarter of the time per iteration. For further insights on fast inverse kinematics solutions, I recommend the work of Lee and Shin who employ a hybrid of analytic and numeric solvers [14].

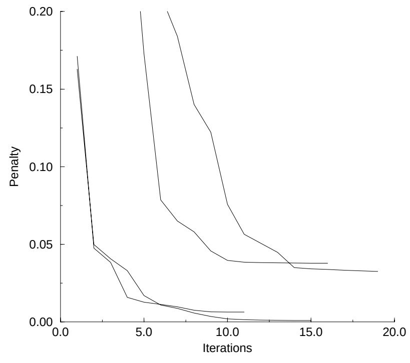

Fig. 2. **Sample from BFGS algorithm.** The overall error resulting from several trials for different motion capture data samples. Postures with larger initial errors tend to converge more slowly and maintain higher errors due to larger elbow position errors.

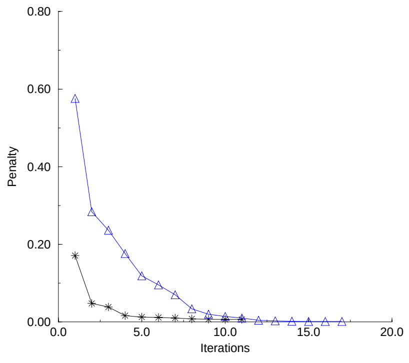

Fig. 3. **Knowledgeable vs. naive starting point for BFGS algorithm.** Plots of different starting points from motion capture (stars) and naive hand-at-side (triangles). As expected, the better initial guess leads to a faster convergence. Because the poor initial guess had a poor elbow position, a lower weighting was given to the elbow goal, resulting in the slightly smaller objective value shown.

B. Control gain tuning for dynamic models

The second problem I investigate is tuning control parameters for a dynamic model. Many controllers in computer animation for active dynamic systems include tuned parameters for stiffness and damping gains. Selection of these parameters can be difficult because the parameter space may be complex, especially in the case where gains are interrelated. One example of gain tuning is the selection of stiffness and damping terms for hip and ankle offsets that maintain balance in a standing model. Dynamic balance may be maintained by controlling the error between the center of mass and the center of support in a dynamic model. Figure 6 shows a schematic of these parameters. In this case, the stiffness and damping gains of the error and derivative error must be chosen carefully because the combined errors and gains are used to adjust the desired joint angles in order to maintain balance, as described by Wooten [24]. I choose these control parameters initially by hand and modify them using an optimization process.

In the problem formulation, dynamic models are constructed using the techniques described by Hodgins and her colleagues [13]. Ground contact constraints, rely on force-driven penalties associated with the feet penetrating the ground. The primary goal in this problem is to minimize the collective error over time between the actual center of mass and the desired center of mass for the simulation. Only solutions that remain standing are considered valid. Solutions where the character falls over are thrown out without evaluation. An additional term for minimizing the body attitude error is added to the evaluation function in order to keep the simulation standing upright. Without this posture goal, as experience showed, there was no restriction on unnatural, leaning poses.

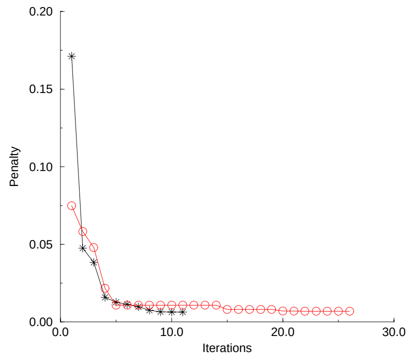

Fig. 4. **Hooke and Jeeves vs. BFGS Algorithms - Knowledgeable starting point.** The same sample configuration using the Hooke and Jeeves (circles) and the BFGS (stars) algorithms. The two algorithms find comparable solutions. Although the Hooke and Jeeves algorithm terminates in more iterations than BFGS, each iteration is computed much more quickly.

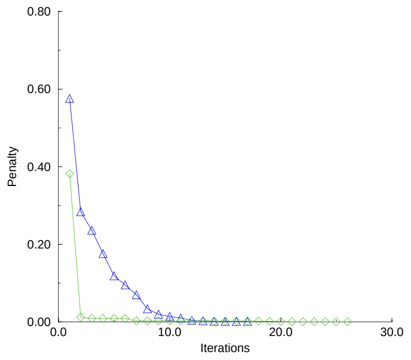

Fig. 5. **Hooke and Jeeves vs. BFGS Algorithms - Naive starting point.** The same sample configuration using the Hooke and Jeeves (diamonds) and the BFGS (triangles) algorithms starting from a naive arm-at-side posture. In this case, the Hooke and Jeeves algorithm converges more quickly from the start because the BFGS algorithm relies on the gradient which is poorly estimated during the large initial jumps in the solution space.

The general form of this gain-tuning optimization problem formulation can be summarized as follows:

Given

Kinematic model

Dynamic model

dynamic equations of motion

Desired joint angles

nominal desired state of standing posture

Find

System Variables γ_i i = 1 to 4

Stiffness for center of mass error in X and Y

Damping for center of mass derivative error in X and Y

Satisfy

Behavior Constraints

Character must remain standing

Ground Constraints

Character must not penetrate the ground

Minimize

Single Objective Archimedian Scheme

$\omega_1 + \omega_2$

$\omega_1 =$ time for center of mass to settle

$\omega_2 =$ final posture orientation error

To solve the balance gain-tuning problem, I implemented two search methods, one stochastic and one gradient. Starting from a poorly tuned but balancing simulation, the gradient-based search should find a local solution that is better according to the objective. However, assuming that the search space of gain values is highly non-linear, a stochastic search seems more appropriate. In this case, I compare simulated annealing to the BFGS algorithm. As in the inverse kinematics solution, the BFGS algorithm uses gradients calculated with finite difference approximations.

A set of different initial conditions and their results are shown for the annealing algorithm in Figure 7. The algorithm found different solutions for several sets of starting gains, implying that the true global minimum was not found. Because the simulated annealing is global, the search

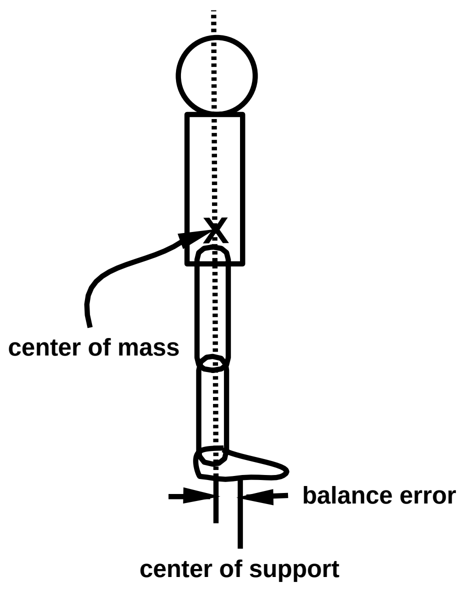

Fig. 6. **Balanced standing.** Balance is maintained by reducing the error between the projected center of mass for the body and the center of support created by the feet. By adjusting the hip and ankle desired angles, this error is controlled.

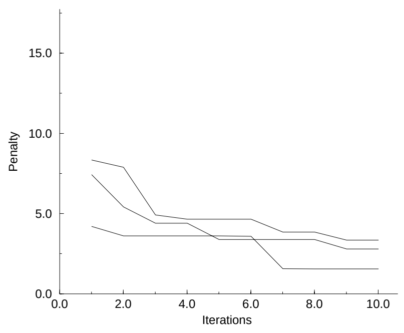

Fig. 7. **Samples from Simulated annealing algorithm.** A set of simulated annealing trials with the best solution found at the end of each stage shown. In most cases, simulated annealing found a reasonable solution in the first stages of cooling, and then made small improvements until the search was halted.

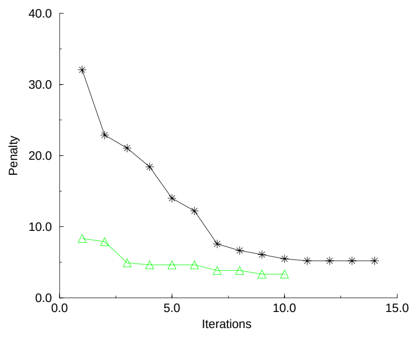

Fig. 8. **Simulated annealing vs. BFGS algorithms.** The same starting gains using the simulated annealing (triangles) and the BFGS (stars) algorithms. The BFGS algorithm made reasonable convergence but overall performed more poorly than simulated annealing.

found better solutions than the gradient method as shown in Figure 8. Although simulated annealing consistently found better solutions for each set of different initial conditions, it did run more slowly than the BFGS algorithm. A slower cooling process and more iterations in the simulated annealing algorithm would likely have led to better results at the cost of a slower process. With simulated annealing taking a longer time to converge than the BFGS approach, a trade-off between computation time and solution accuracy arises.

Overall, validation of the optimization algorithm, constructing experiments, and comparing alternative solvers like the examples described here can ensure that an optimization routine is robust, and that the solutions found are reasonable and efficient. Constructing and calculating error metrics for objective functions remain the most fragile pieces of the problem formulation. Several objective functions may need to be tested before a satisfactory choice is made.

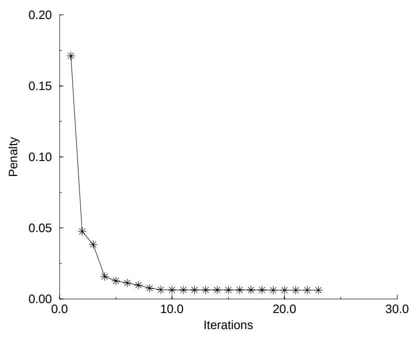

Fig. 9. **Stopping criterion too strict.** While this function does show a nice convergence for the objective function, the stopping criterion is too strict, causing several iterations to be calculated without improvement before the solver halted. This wastes calculation time and resources, indicating a looser stopping criterion should be chosen.

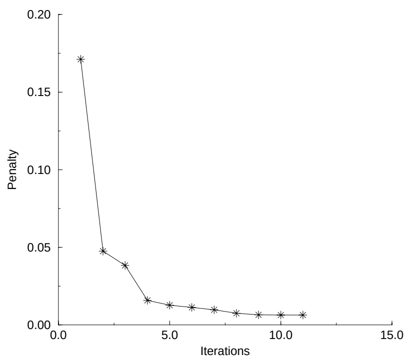

Fig. 10. **Tuned Optimizer.** This case was modified from the original in two ways: the stopping condition was tuned to recognize convergence more quickly, and the gradient approximation was started with a tighter step size to promote quicker gradient calculations. The solution maintains similar characteristics, but the solver runs and halts more quickly.

VI. Conclusions

This report includes a general framework for considering and formulating an optimization problem and a breakdown of possible techniques to solve the problem. Published works and detailed formulations for two problems bind the techniques to computer animation applications.

In practice, many problems can be formulated in a manner that makes them too difficult to be solved using numeric optimization. However, insights about the solution space found by user exploration help establish reasonable expectations and may suggest reformulating the problem at times. A good sampling of the solution space will reveal important factors such as whether the space is rough and discontinuous or smooth. Choosing good initial conditions and specifying small search ranges can make an optimization problem much easier. Also, carefully breaking a problem into a series of smaller problems can sometimes make a finding a solution more manageable. However, in some cases, numeric optimization is simply not the right answer because some spaces are too sparse to search and some do not contain good solutions. In these cases, a closer look at the specific application may help reveal a simpler optimization approach that can be formulated.

Tuning parameters such as error tolerances and termination requirements is important for efficiency. Figure 9 displays a plot of the untuned solver from the IK solution described. After the majority of the improvement was made in the first few iterations, the BFGS algorithm ran without much improvement for several iterations. By tuning the tolerances on the error allowed, the solver halted more quickly. Also, the algorithm spent much of its time calculating the gradient. A better step-size for the Riddler algorithm improved the performance. Figure 10 reflects the tuned optimizer which has the same characteristics as the original, but runs in about one third of the time.

In this report I introduce basic numeric techniques and recommend further investigation into more advanced approaches. In general, optimization is a complete field unto itself. The animation community is interested in using optimization methods as tools for problems but this toolset is ever-changing. Keeping a current perspective on the advances in the area of optimization ensures that an animator is using the most powerful and appropriate methods available when solving computer animation optimization problems.

References

[1] Brian Anderson and John Moore. *Optimal control: linear quadratic methods.* Prentice Hall, New Jersey, 1990.

[2] J. Baillieul and D. Martin. Resolution of kinematic redundancy. In *Proceedings of Symposia in Applied Mathematics*, pages 49–89. American Mathematical Society, 1990.

[3] Bobby Bodenheimer, Charles Rose, Seth Rosenthal, and John Pella. The process of motion capture: Dealing with the data. In *Computer Animation and Simulation '97*, pages 3–18. Eurographics, Springer-Verlag, September 1997.

[4] Lynne Shapiro Brotman and Arun N. Netravali. Motion interpolation by optimal control. In *Proceedings of SIGGRAPH '88*, volume 22, pages 309–315. ACM SIGGRAPH, August 1988. Held in Atlanta, Georgia, 1-5 August 1988.

[5] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. In *Proceedings of SIGGRAPH '89*, volume 23, pages 233–242. ACM SIGGRAPH, July 1989. Held in Boston, Massachusetts, 31 July - 4 August 1989.

[6] Michael F. Cohen. Interactive spacetime control for animation. In *Proceedings of SIGGRAPH '92*, volume 26, pages 293–302. ACM SIGGRAPH, July 1992. Held in Chicago, Illinois, 26-31 July 1992.

[7] Peter Dyer and Stephen McReynolds. *The computation and theory of optimal control.* Academic Press, New York, 1970.

[8] Michael Gleicher. Motion editing with spacetime constraints. In *Proceedings of 1997 Symposium on 3D Graphics*, pages 139–148. ACM SIGGRAPH, April 1997. Held in Providence, Rhode Island, 27 - 30 April 1997.

[9] Michael Gleicher. Retargeting motion to new characters. *Proceedings of SIGGRAPH 98*, pages 33–42, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[10] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of Muscle-Actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH '95*, pages 63–70. ACM SIGGRAPH, August 1995. held in Los Angeles, California, 06-11 August 1995.

[11] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. *Proceedings of SIGGRAPH 98*, pages 9–20, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[12] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series on Computer Graphics, pages 153–1–62. ACM SIGGRAPH, Addison Wesley, August 1997. held in Los Angeles, California, 03-08 August 1997.

[13] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of SIGGRAPH '95*, pages 71–78. ACM SIGGRAPH, August 1995. Held in Los Angeles, California, 06-11 August 1995.

[14] Jehee Lee and Sung Yong Shin. A hierachical approach to interactive motion editing for humanlike figures. In *Proceedings of SIGGRAPH '99*, pages 39–48. ACM SIGGRAPH, August 1999. Held in Los Angeles, California, 08-13 August 1999.

[15] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. In *Proceedings of SIGGRAPH '93*, pages 343–350. ACM SIGGRAPH, August 1993. Held in Orlando, Florida, 24-29 July 1994.

[16] Dinesh Pai. Programming anthropoid walking: Control and simulation. Technical Report 90-1178, Cornell Computer Science, 1990.

[17] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* Cambridge University Press, New York, 1994.

[18] G. Reklaitis, A. Ravindran, and K. Ragsdell. *Engineering Optimization: Methods and Applications.* John Wiley and Sons, New York, 1983.

[19] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of SIGGRAPH '96*, pages 147–154. ACM SIGGRAPH, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996.

[20] Karl Sims. Evolving 3d morphology and behavior by competition. In *Artificial Life IV*, pages 28–39, 1994.

[21] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In *Proceedings of SIGGRAPH '93*, pages 335–342. ACM SIGGRAPH, August 1993. Held in Orlando, Florida, 24-29 July 1994.

[22] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controllers. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 225–234, August 1990.

[23] Andrew Witkin and Michael Kass. Spacetime constraints. In *Proceedings of SIGGRAPH '88*, pages 159–168. ACM SIGGRAPH, August 1988. Held in Atlanta, Georgia, 1-5 August 1988.

[24] Wayne L. Wooten. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans.* Ph.D. Thesis, Georgia Institute of Technology, 1998.