Spacetime Constraints Revisited

J. Thomas Ngo
Graduate Biophysics Program
*Harvard University**

Joe Marks
Cambridge Research Lab
Digital Equipment Corporation†

Abstract

The Spacetime Constraints (SC) paradigm, whereby the animator specifies what an animated figure should do but not how to do it, is a very appealing approach to animation. However, the algorithms available for realizing the SC approach are limited. Current techniques are local in nature: they all use some kind of perturbational analysis to refine an initial trajectory. We propose a global search algorithm that is capable of generating multiple novel trajectories for SC problems from scratch. The key elements of our search strategy are a method for encoding trajectories as behaviors, and a genetic search algorithm for choosing behavior parameters that is currently implemented on a massively parallel computer. We describe the algorithm and show computed solutions to SC problems for 2D articulated figures.

CR Categories: I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*. I.2.6 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*heuristic methods*. I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*animation*. I.6.3 [**Simulation and Modeling**]: Applications.

Additional Key Words: Genetic algorithms, evolutionary computation, massive parallelism.

1 Introduction

The Spacetime Constraints (SC) formulation was proposed in 1988 by Witkin and Kass as a powerful paradigm for indirectly controlling the physically realistic motion of articulated figures ("creatures") for animation purposes [17]. The essence of the SC approach is to have the animator specify what the animated creature should do, and the computer determine how the creature should do it. Specifically, the animator defines:

- the physical structure of the creature;
- the actuators that control the creature's internal configuration; and
- criteria for evaluating the creature's motion.

*Present address: Interval Research Corporation, 1801-C Page Mill Road, Palo Alto, CA 94304. Email: ngo@interval.com.

†One Kendall Square, Building 700, Cambridge, MA 02139. Email: marks@crl.dec.com.

The computer is left with the task of computing a physically realistic trajectory for the creature that is optimal according to the animator-supplied criteria.

The computer's role in this paradigm is quite challenging.[1] Finding globally optimal solutions to SC problems is hard for two main reasons [13]:

- Multimodality—even when the problem is suitably discretized, there are an exponential number of possible trajectories that a creature can follow, many of which may be locally optimal or near optimal.
- Search-space discontinuities—a small change in the behavior of a creature's actuators can lead to a large change in its trajectory.

Current algorithmic techniques for SC problems sidestep the difficulties of global optimization entirely by settling for some form of local optimization: typically an initial trajectory is perturbed in some way until it is locally optimal [2, 17, 3]. In this paper we propose a global search algorithm that fulfills a complementary need by generating multiple near-optimal trajectories for SC problems from scratch. The key elements of our search strategy are a method for encoding trajectories as sets of stimulus-response behavior rules [14, 16, 13], and a genetic algorithm (GA) [10, 8, 5] for choosing behavior parameters that is currently implemented on a massively parallel SIMD computer.

2 Algorithm

Our algorithm can be described in summary as follows:

- A dynamics module (§2.1) simulates a physically correct virtual environment in which the effects of creature behaviors may be tested by trial and error.
- A behavior module (§2.2) generates such behaviors using a parameterized algorithm that is based on the concepts of stimulus and response [14].
- A search module (§2.3) uses a genetic algorithm to choose values for the stimulus and response parameters that will generate near-optimal behaviors according to the evaluation criteria for the given SC problem.

[1] This is not to suggest that the animator's role has been trivialized completely—far from it! Specifying appropriate evaluation criteria for a creature's motion is a challenging problem, as is the task of reducing a complex animation problem to a concatenated sequence of related SC problems [3].

2.1 Physical simulator

Like Hahn's [9], our physical simulator employs forward *dynamics*, but treats an articulated figure as an autonomously deforming object without internal degrees of freedom. The deformations are produced *kinematically* by the creature's stimulus-response control algorithm (§2.2). The principal advantage of this approach is dramatically reduced CPU cost.[2] We find that in practice there is little sacrifice in terms of physical realism even though torques about the joints are never computed explicitly. If torque limits were to be important for a particular application, a penalty for excessive torques could be added to the evaluation criteria; the torques can be computed using inverse dynamics [11]. However, we have found that simply limiting the rate at which a creature's internal degrees of freedom can accelerate—something that can be specified trivially in our behavior-based representation—is enough to produce visually reasonable behavior.

Friction is taken to be static when one joint is in contact with the floor, and slippage is proportional to contact force when two joints touch. Collision and contact forces are treated essentially[3] as in Baraff [1]. We specialized Baraff's elegant treatment to two dimensions, then extended it to accommodate autonomous deformations. For a rigid polygon in 2D interacting with a flat 1D floor, the contact-force computation is much simpler than in the 3D case, because the number of contact points is limited to two. The extension to accommodate autonomous deformations is straightforward as well. The key to Baraff's technique is that the relative normal acceleration ($\ddot{\chi}_i$) for a given contact point i turns out to be a linear function of the vector of normal contact forces ($\vec{f}$). The relation remains linear after adding a contribution to $\ddot{\chi}_i$ that takes into account the creature's deformation, so the procedure for satisfying the "non-penetration" and "non-stickiness" constraints ($\ddot{\chi}_i \geq 0$ and $f_i \geq 0$) remains essentially unchanged.

2.2 Stimulus-response representation

Trial behaviors to be tested in the simulator (§2.1) are generated by the stimulus-response (SR) control algorithm. This algorithm neither learns nor plans; rather, it causes the creature to execute instinctive reflexes triggered by conditions sensed in its virtual environment. The conditions are called *stimulus functions* and the reflexes are called *responses*; and their parameters remain fixed for the duration of a trial behavior. The search module (§2.3) finds values for both the stimulus parameters and the response parameters essentially by trial and error. The use of these SR parameters in lieu of a more conventional time series of configurations or forces may be the most important factor in the success of our approach to SC problems.

A *response* is a prescription for changing the creature's shape smoothly. It consists of a time constant and a complete set of target values for the creature's internal angles. Applying the response for one time step means iterating through one time step of the critically damped equation of motion

$$\tau^2 \ddot{\theta}_i + 2\tau \dot{\theta}_i + (\theta_i - \theta_i^0) = 0$$

for each of the internal angles, where τ is the time constant, θ_i^0 is the target value of the internal angle, and θ_i is its actual value. The effect of this equation of motion is to cause the real shape of the creature, given by $\{\theta_1, \theta_2, \ldots, \theta_N\}$, to approach the target shape $\{\theta_1^0, \theta_2^0, \ldots, \theta_N^0\}$ smoothly, even when the target shape changes abruptly due to a switch from one response to another.

To define a stimulus function we must introduce the concept of a sense variable. A *sense variable* is some real-valued function of the physical environment. Our standard list includes:

- proprioceptive senses—each joint angle;
- tactile senses—the force exerted by each rod endpoint on the floor (and *vice versa*);
- kinesthetic sense—the vertical velocity of the center of mass; and
- position sense—the vertical position of the center of mass relative to the floor.

A *stimulus function* is a scalar function defined over sense space. If the sense variables are $\{v_1, v_2, \ldots, v_V\}$, then the expression that we use for a stimulus function is:

$$W \left\{ 1 - \max_{j=1}^{V} \left[\lambda_j (v_j - v_j^0) \right]^2 \right\},$$

where the v_j^0 and λ_j are parameters determined by the search module (§2.3), the weight W is:

$$W = \sum_{j=1}^{V} \log \left(\frac{\lambda_j}{\lambda_j^{\min}} \right),$$

and $\lambda_j^{\min}$ is the predetermined smallest permissible value of λ_j (§2.3). (In practice all of these constants are normalized so that the sense variables v_j fall between 0 and 1.) The locus of points in sense space for which the stimulus-function expression is positive—its *sensitive region*—is a hyper-rectangle with dimensions $\{2/\lambda_1, 2/\lambda_2, \ldots, 2/\lambda_V\}$ centered at $\{v_1^0, v_2^0, \ldots, v_V^0\}$.

The set of SR parameters for a creature consists of an array of SR pairs (10 pairs in all the tests we have run to date). In the following pseudocode, which describes how an SR array is used to generate behavior, the state variables are the creature's physical state and a pointer to the active response:

Initialize creature state from SC problem description
Activate response 0
for t = 1 **to** T
    Determine deformation for time t from active response
    Simulate resulting *dynamics* for time t
    Measure sense variables from the environment
    Identify highest-valued stimulus function
    Activate corresponding response if stimulus positive
end for

The final step may or may not change which response is active; a change is made only if the highest-valued stimulus function is positive. Thus, a response is typically active for several consecutive time steps and produces coherent motion.

[2] Physical simulation is the CPU-intensive portion of our approach.

[3] A modification was necessitated by the SIMD architecture of our machine, because of which we implemented a fixed-timestep integrator with analytic recomputation after a mid-timeslice collision, rather than a more accurate variable-timestep integrator with instantaneous handling of the impulsive forces that arise from collisions.

2.3 Genetic algorithm

In §2.1 and §2.2 we described what a creature would do given a structural specification, initial conditions, and values for its stimulus and response parameters. In §3 we will supply examples of how the resulting behaviors might be evaluated quantitatively. Taken together, these ingredients define a scalar function of the SR parameters. It remains to show how to find near-optimal values of this function.

Our search procedure is a parallel genetic algorithm (GA) written in C* on a Thinking Machines CM-2 with 4096 processors. In a GA, a population of candidate solutions is subjected to a procedure that simulates biological evolution. Each candidate solution—each *genome*, in GA parlance—has some probability of being mutated, recombining with another genome, and dying, based on its value. In our implementation, each processor is responsible for evaluating one genome per generation. Our GA is described below in pseudocode:

do parallel
    Randomize genome
end do
for generation = 1 **to** number_of_generations
    do parallel
        Evaluate genome
        Select mate from another processor
        Cross genome with mate
        Mutate genome
    end do
end for

In this section we shall explain the component operations of this genetic algorithm: *Randomize*, *Select mate*, *Cross*, and *Mutate*. Although these operations have worked without modification for all of the test cases shown here and elsewhere [13], they should not be considered optimal.

Symbol	Description	Lo	Hi	Distribution
τ	Time constant	$2\Delta t$	$\frac{1}{4}T$	Logarithmic
θ_i^0	Target angles	$\theta_i^{\min}$	$\theta_i^{\max}$	Uniform
v_j^0	Stimulus centers	-0.5	1.5	Uniform
$2/\lambda_j$	Stimulus extents	0.4	4	Logarithmic

Table 1: The parameters that comprise one SR pair in the genome, their typical ranges, and their probability distributions in the initial random population. The timestep size and total simulation time are Δt and T, respectively; and the quantities $\theta_i^{\min}$ and $\theta_i^{\max}$ are lower and upper bounds on the joint angles taken from the structural specification of the creature.

Randomization Values for all of the parameters in the genome are chosen at random in the initial population, with the probability distributions given in Table 1. We then employ a form of hill-climbing to enrich the initial gene pool: after evaluating this initial population, we mutate and re-evaluate each solution four times, and on each processor choose the best out of five. Because of our highly specialized mutation operator (see below), this has the effect of producing a fairly non-random population that is skewed in favor of multi-step behaviors.

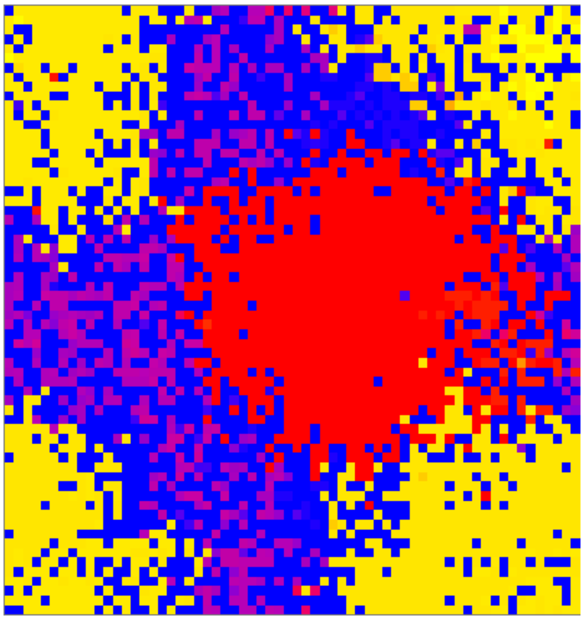

Figure 1: Visualization of a GA population. The display consists of 64×64 pixels, each one corresponding to a processor in the CM-2. Each pixel is colored according to the value of the solution stored at the corresponding processor: the colors range from dark blue for the worst solutions to bright yellow for the best ones. In this mature population two relatively homogeneous regions ("colonies") have emerged from a background of mediocre solutions; the corner areas form one region on the toroidal grid.

Mate selection This is the only step in the algorithm that requires interprocessor communication, and it is also the only stage at which better solutions are given an advantage. In the simplest GA's, mating between any two individuals is possible. To prolong the diversity of the population—this helps cope with multimodality—it is common practice to impose a geographic distribution on the population and to permit only local mating. In our implementation, the processors are laid out on an imaginary 64×64 toroidal grid; this scheme is especially easy to achieve on the CM-2. To choose a mate, each processor performs a random walk of (typically) 10 steps on the grid, and identifies the best solution encountered. If the best solution encountered is itself, it does not mate. (Thus, a solution that is best in its neighborhood is protected from crossover and mutation; this is a form of *elitism* [7].) Otherwise, it fetches a copy of the selected mate's genome in preparation for crossover.

This local mating scheme causes good "genes" (solution fragments) to diffuse slowly through the population, leading to the formation of "colonies" of similar solutions. Figure 1 depicts a population that arose in one of our experiments. As a colony spreads, more processors are put to work on variations of the superior solution that spawned the colony. But because the spreading is not instantaneous, suboptimal solutions still get some processing time. When one colony dominates (a typical outcome when genetic algorithms of this type are left to run indefinitely), the population is said to have converged.

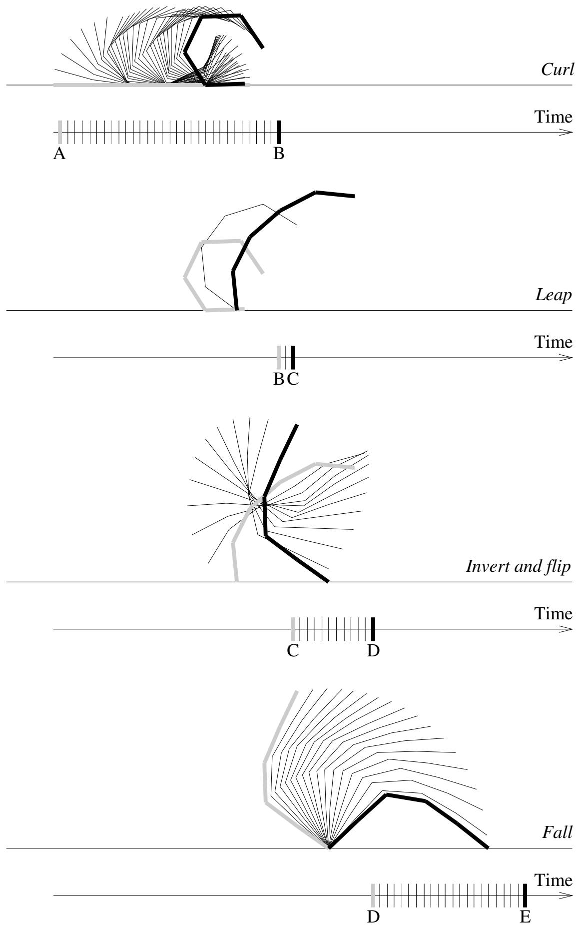

Figure 2: Five-Rod Fred moves to the right by curling and leaping. Since his initial configuration is symmetric, his curling action must be asymmetric in order to generate net rightward motion. Momentum is essential in permitting a forward leap. Inversion in the air is necessary to delay landing long enough to fall forward instead of backward. **Note:** In this and subsequent figures we have arbitrarily broken the motion into easily perceived phases. This is only for presentation purposes, and is not indicative of any aspect of the algorithm. Ticks along the horizontal time axis show which time slices are drawn, and in what linestyles. The first, intervening, and last frames in each phase are drawn with thick grey, thin black, and thick black lines, respectively. An italicized phrase at the lower right of each figure describes the corresponding action.

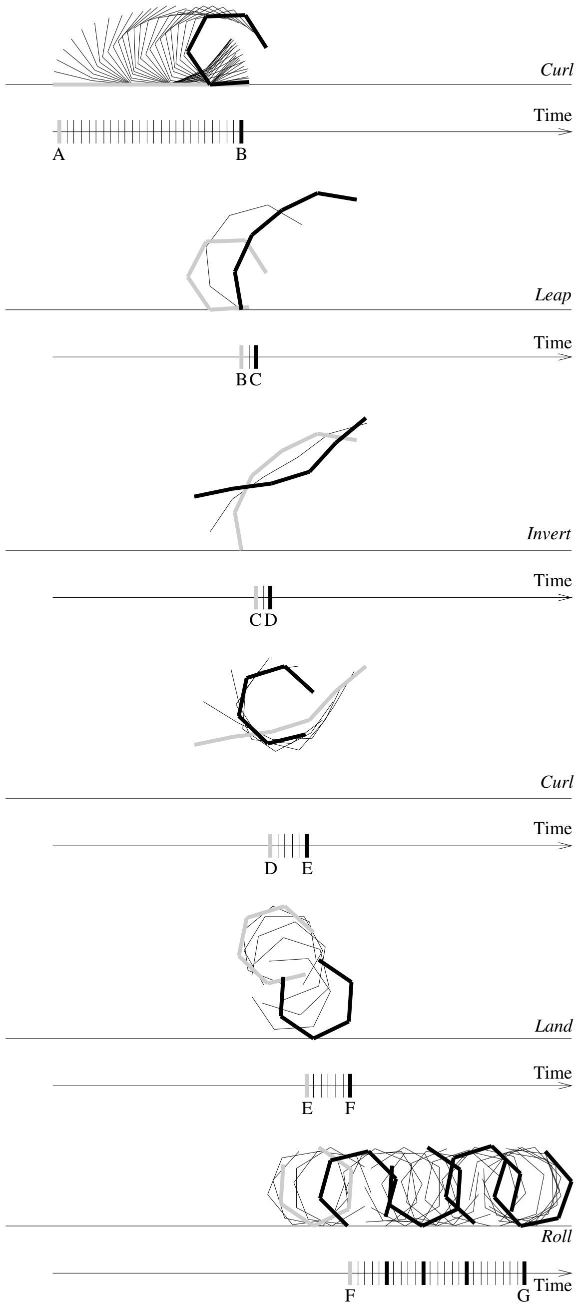

Figure 3: Fred rolls on landing. Because his curled configuration is a hexagon and not a circle, Fred bounces as he rolls.

Crossover The crossover operators used in the GA as it was originally described [10] are based on a relatively literal interpretation of the biological metaphor: the bit representations of the parameters in the genomes are placed end-to-end in linear fashion, and crossover operations call for snipping two genomes at analogous locations along the bit string, and swapping one pair of ends.

In the SR representation, a linear layout would not be very meaningful, and we know in advance that certain groups of parameters should migrate together between genomes. Therefore our crossover operator is tailored to treat the genome in a more structured fashion. Given the probabilities that we use, a hybrid might be constructed from two genomes (of ten SR pairs each) as follows: six intact SR pairs are taken from the mate and two from the original genome; one new SR pair is constructed by taking a stimulus and a response from each parent, respectively; and one new SR pair is constructed by uniform crossover [15], *i.e.*, each number in the SR data structure is copied at random from one or the other parent. The precise probabilities that we use in performing the crossover are not critical; we have not tried to optimize them because the values we use currently are probably irrelevant for fu-

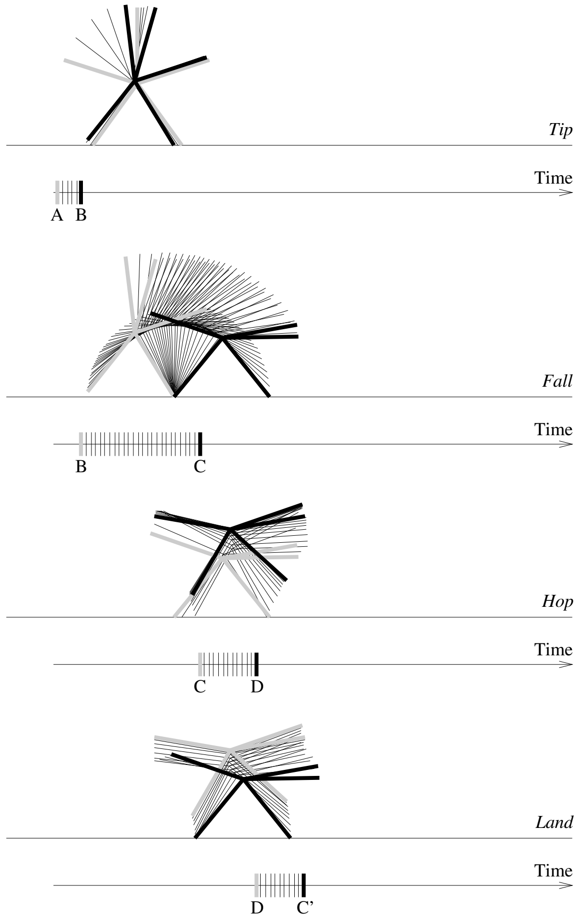

Figure 4: Mr. Star-Man canters on two side limbs. The C–D–C’ sequence is repeated cyclically. Were Mr. Star-Man to have human-like strength and structure he would not be able to walk this way for very long, since his arm would get tired. This is one situation in which penalties for excessive torques might influence the range of behaviors discovered.

ture implementations of our general approach (§4). The key point is that the crossover operation must be tailored to the problem to get good performance.

Mutation Our mutation operator [13] is also tailored specifically for the SR representation. One SR pair is subjected to *creep* [5]; *i.e.*, each of the parameters in that SR pair is changed by a small amount. Another SR pair is randomized from scratch, with one stipulation that turns out to be quite important in coping with a large number of sense variables: that at least one corner of the sensitive region (§2.2) of the newly generated stimulus function be guaranteed to coincide with the original sense-space trajectory. Without this restriction, freshly generated stimulus functions tend either to dominate the trajectory or not to modify it at all.

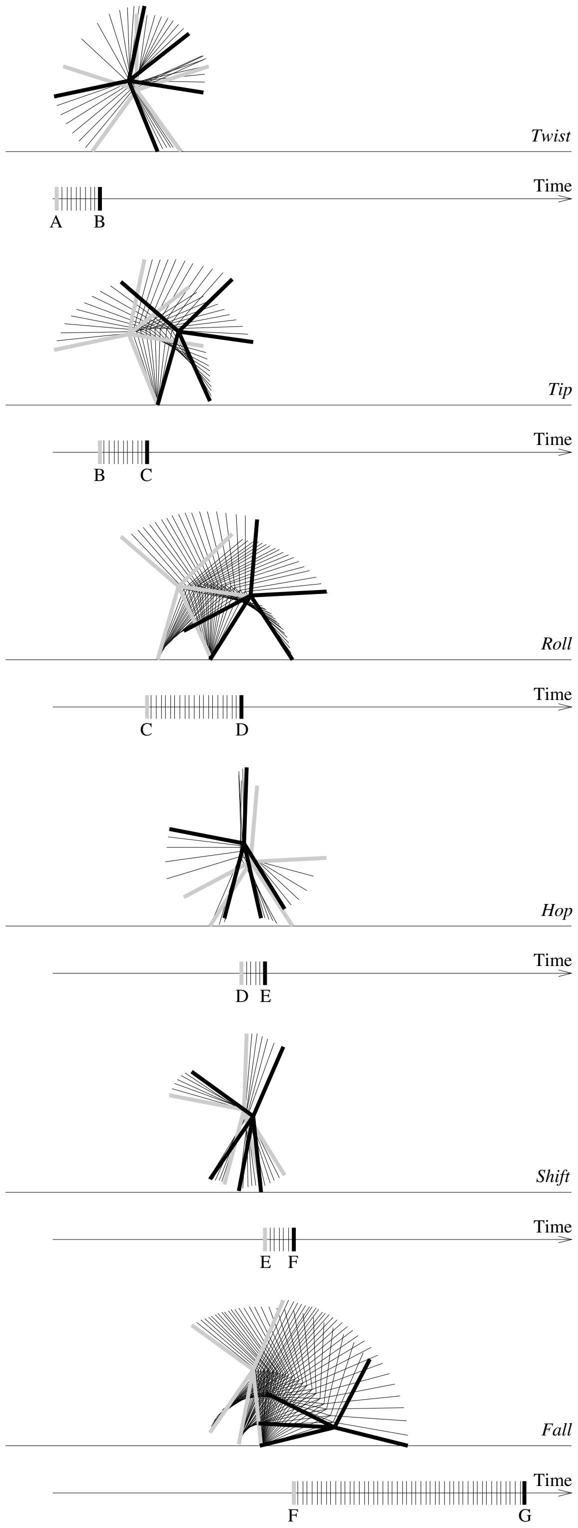

Figure 5: Mr. Star-Man begins to cartwheel, but when his time is nearly up, he falls to the finish line. The subtle posture shifts in A–B and E–F are critical to the motion.

3 Results

In an earlier suite of tests involving unbranched articulated figures [13], we showed that the algorithm could cope with two-point

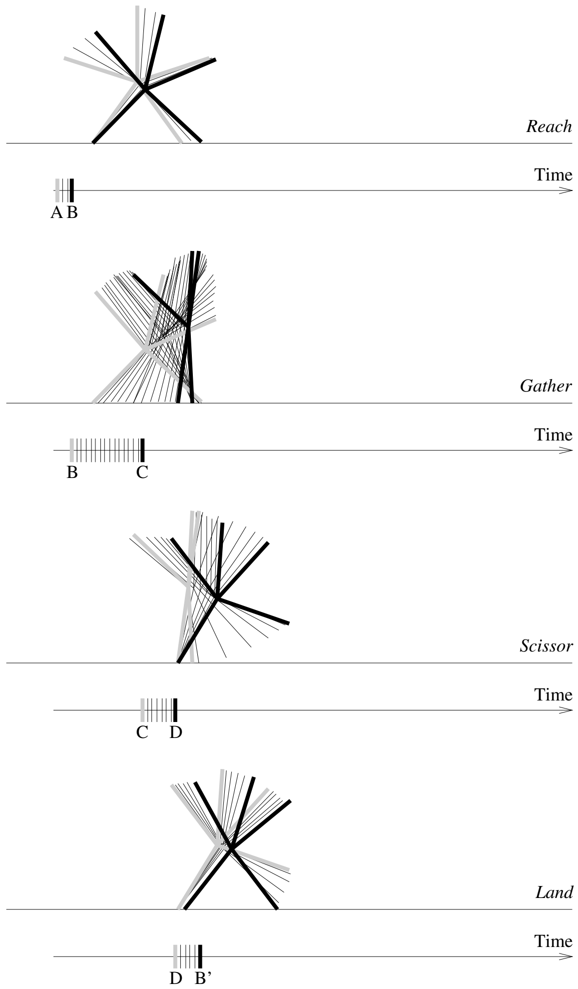

Figure 6: Mr. Star-Man shuffles. The B–C–D–B' sequence is repeated cyclically. Star-Man's stride length is limited by how long his right "foot" can stay off the ground during the step marked *Scissor*. His "arm" helps prolong this step by swinging down, offsetting the upward momentum contribution by his "leg."

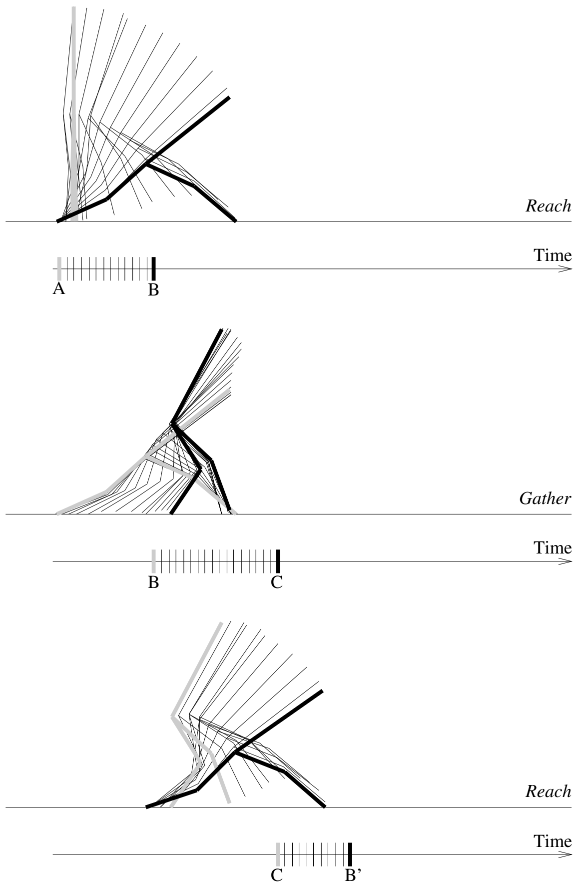

Figure 7: Beryl Biped skips. The B–C–B' sequence is repeated cyclically. Skipping differs from walking; the foot that starts out in the back never moves to the front. Balance is less of an issue in this type of skipping than in walking because intermediate postures are statically stable. However, this skipping is evidently not as fast as walking can be.

boundary conditions as well as one-point boundary conditions with "athletic" evaluation functions (jump height and walking distance). The present test suite is a further exploration of the "walking" problem that involves bigger, more complex articulated figures, most of which are branched.

We present results for three selected runs. In the first two cases, the evaluation function was proportional to the net horizontal distance covered by the creature's center of mass in the given time. The farther the creature's center of mass moved to the right, the better the trajectory—regardless of how the movement was accomplished. In the third case, the evaluation function had to be modified slightly to obtain the desired behavior. In each case, approximately 30 to 60 minutes of elapsed time on the CM-2 was required to compute each solution shown.[4]

The first problem involves Five-Rod Fred, a creature comprised of five equal-length rods, linked consecutively. The middle rods are of equal mass, but the terminal rods are five times heavier. Each joint allows its pair of connected rods to be at most 30° from collinear. We constructed this creature in the expectation that it would behave like an inchworm. To our surprise, 64 generations of evolution produced the solution depicted in Figure 2. After 100 generations, this behavior was improved by the addition of a rolling phase at the end of the motion (Figure 3).

Fred's final behavior is generated by just five of the ten available SR pairs, which is not unusual for the behaviors described here. Initially, two SR pairs produce the asymmetric curling motion shown in the first panel of Figure 3. Two more SR pairs produce the leaping motion. Finally, a single SR pair causes Fred to adopt an inversely curled configuration for the remainder of the trajectory.

Mr. Star-Man is another five-rod creature, but one with a different, branched topology. All of his rods are of equal length and mass. Joint-angle ranges are defined with respect to Star-Man's "torso": each "limb" rod is confined to one of the quadrants defined relative to his top "torso" rod. The trajectory in Figure 4

[4] The quoted times are for 5 rods, 50 time steps, and 100 generations.

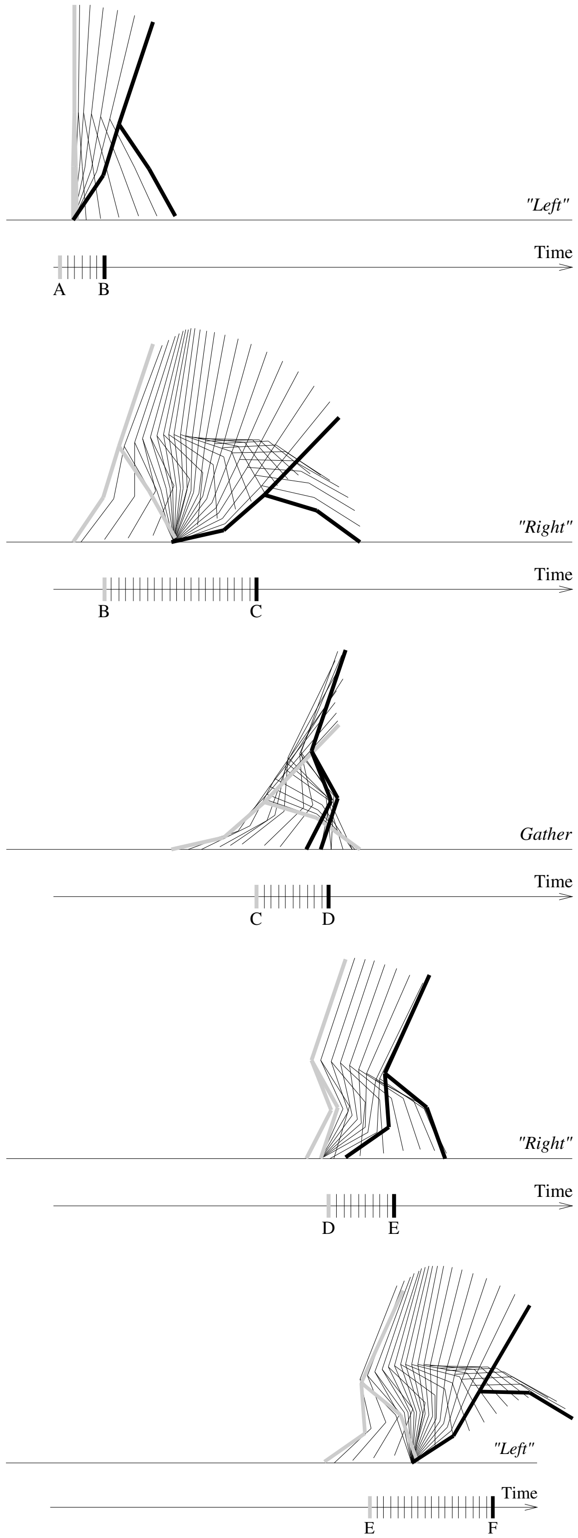

Figure 8: Beryl walks, but one step (C–D–E) is really a skip. The GA/SR combination frequently produces hybrid gaits of this type.

depicts a behavior that evolved after 20 generations: Star-Man tips

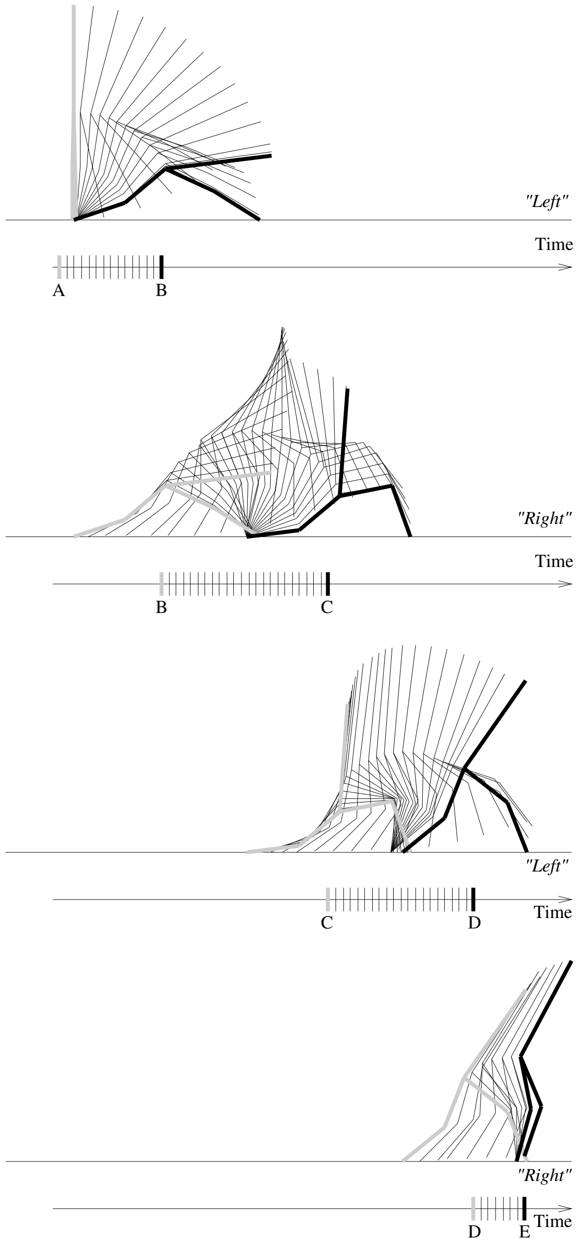

Figure 9: Beryl walks. How to modify the evaluation function to reward grace as well as distance remains an open question.

over on his side, and then employs a sideways cantering motion. This motion is cyclic; the hop-land sequence (C–D–C') is repeated several times. By generation 37 a new behavior had evolved—the start of a cartwheel maneuver followed by a fall to the right (Figure 5). The best behavior after 94 generations was yet a third strategy that involved sideways shuffling (Figure 6).

Beryl Biped is a headless 2D humanoid with a rigid torso, jointed legs, point feet, and rod masses of human proportion. Despite her limitations, Beryl learns human-like locomotion. To elicit these behaviors, we had to change the evaluation function. When center-of-mass translation was used as the measure of progress, Beryl adopted the short-term strategy of plunging headlong into the ground (not shown). When the reference point was changed from the center of mass to the midpoint between her two feet, she still tried falling forward, but in 100 generations she discovered the more tactically sound gaits depicted in Figures 7–9. Thus, like any optimization procedure, the genetic algorithm will fail to find good optima if the evaluation function is sufficiently "deceptive," but simple changes to the evaluation function can often restore acceptable behavior.

4 Conclusions and future work

We have presented an effective algorithm for 2D SC problems involving articulated figures. The trajectories computed by our algorithm differ qualitatively from those that would be produced by existing local-search techniques: they are complex, varied, multi-staged, and sometimes far from obvious. Our work differs from previous global-search and learning approaches to articulated-figure motion control [6, 4, 12] in its adherence to physical law, the nature of the articulated figures being considered, and the generality of the problem statement, respectively.

Developing an algorithm of this type requires numerous decisions regarding the design of mutation and crossover operators and the assignment of parameter values. It is difficult to be sure that one has chosen optimally, since doing so is itself a combinatorial search problem! However, few of the choices are critical. In this paper and elsewhere [13] we have described what we believe are the important choices to get right.

Our current work is directed towards gaining a better understanding of how the algorithm works and making more use of its present capabilities. In particular we are investigating other stimulus-response representations, ways in which a user can influence the GA interactively, alternative search algorithms, and editing techniques whereby composite trajectories can be formed by splicing together precomputed behaviors. In the future, we hope to extend our techniques to work with 3D articulated figures.

5 Acknowledgments

We thank Tom Cheatham for use of the CM-2 at Harvard. Carter Cornwall, Dave Davis, Steven Salzberg, Stuart Shieber, and the members of the Harvard Animation Group, the MIT Leg Lab, and the Cambridge Seminar on Natural and Artificial Computation all provided useful advice and suggestions. Adam Ginsburg wrote support software for the project. JTN is grateful for a Graduate Fellowship from the Fannie and John Hertz Foundation. This work was supported in part by an NSF grant to Martin Karplus.

References

[1] D. Baraff. Coping with friction for non-penetrating rigid body simulation. *Computer Graphics*, 25(4):31–40, July 1991.

[2] L. S. Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

[3] M. F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[4] Y. Davidor. A genetic algorithm applied to robot trajectory generation. In L. Davis, editor, *Handbook of Genetic Algorithms*, chapter 12, pages 144–165. Van Nostrand Reinhold, New York, 1991.

[5] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[6] H. de Garis. Genetic programming: Building artificial nervous systems using genetically programmed neural network modules. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 132–139, Austin, Texas, June 1990.

[7] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Algorithms*. PhD thesis, University of Michigan, 1975.

[8] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1988.

[9] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[10] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[11] J. Luh, M. Walker, and R. Paul. On-line computational scheme for mechanical manipulators. *Trans. ASME, J. Dynamic Systems, Measurement, and Control*, 102:69–76, 1980.

[12] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 796–802, Menlo Park, California, 1990. American Association for Artificial Intelligence.

[13] J. T. Ngo and J. Marks. Physically realistic trajectory planning in animation: A stimulus-response approach. Technical Report TR-21-92, Center for Research in Computing Technology, Harvard University, October 1992.

[14] B. F. Skinner. *The Behavior of Organisms; An Experimental Analysis*. The Century Psychology Series. D. Appleton-Century, New York, London, 1938.

[15] G. Syswerda. Uniform crossover in genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, 1989.

[16] J. Wilhelms and R. Skinner. A "notion" for interactive behavioral animation control. *IEEE Computer Graphics and Applications*, 10(3):14–22, May 1990.

[17] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.