Motion-Synthesis Techniques for 2D Articulated Figures

Alex Fukunaga, Harvard University
Lloyd Hsu, Harvard University
Peter Reiss, Harvard University
Andrew Shuman, Harvard University
Jon Christensen, Harvard University
Joe Marks, Digital Equipment Corporation
J. Thomas Ngo, Harvard University

Abstract

In this paper we extend previous work on automatic motion synthesis for physically realistic 2D articulated figures in three ways. First, we describe an improved motion-synthesis algorithm that runs substantially faster than previously reported algorithms. Second, we present two new techniques for influencing the style of the motions generated by the algorithm. These techniques can be used by an animator to achieve a desired movement style, or they can be used to guarantee variety in the motions synthesized over several runs of the algorithm. Finally, we describe an animation editor that supports the interactive concatenation of existing, automatically generated motion controllers to produce complex, composite trajectories. Taken together, these results suggest how a usable, useful system for articulated-figure motion synthesis might be developed.

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: Spacetime constraints, controller synthesis, banked stimulus-response (BSR) controllers, stochastic optimization, evolutionary computation.

1 Introduction

Automatic motion synthesis for articulated figures is the problem posed by the Spacetime Constraints (SC) paradigm for animation [19]. In this paradigm, the animator specifies only the physical structure of an articulated figure and quantitative criteria for success in a desired task. The computer must compute physically realistic motion for the figure that is near optimal[1] with respect to the task criteria.

Early motion-synthesis algorithms used local optimization to refine initial figure trajectories by making them more compliant with physical law, or by improving the motion with respect to the task criteria [1, 19, 2]. However, local optimization has inherent limitations for this problem: it is usually confounded by the discontinuities and local optima found in the search space of a typical SC problem, and it leaves primary responsibility for constructing coarse initial trajectories with the human animator.

Recently, a new approach to the motion-synthesis problem has been proposed. In this approach, the goal is not to compute the figure's trajectory directly, but instead to generate a *motion controller* that, when executed, will produce the desired motion [10, 9, 16]. In any particular embodiment of this approach, two broad and nearly independent choices must be made:

1. how the motion controller is to be represented; and
2. how the space of possible controllers is to be searched.

Van de Panne and Fiume [16] have described one such system in which the motion controller is a sensor-actuator network (SAN). In a SAN, the actuators' responses are interdependent nonlinear functions of the figure's physical state variables. Van de Panne and Fiume search the space of possible controllers in two stages. The first stage is a random generate-and-test procedure, and the second stage effects a subsequent refinement by simulated annealing or stochastic gradient ascent.

In contrast, Ngo and Marks employ a bank of mutually independent controllers called stimulus-response (SR) rules. Information from the physical environment is used principally to determine which rule is active at any given time in the physical simulation. We refer to this as a banked stimulus-response (BSR) controller to distinguish it from the SAN. Ngo and Marks [10, 9] search within the space of possible BSR controllers using a massively parallel genetic algorithm (GA).

In this paper we focus on three major shortcomings of the system described by Ngo and Marks:[2]

1. The search algorithm is expensive: finding a simple motion controller for a five-rod articulated figure can take 30–60 minutes on a 4096-processor CM-2 Connection Machine.
2. There is no mechanism to influence the search algorithm: the algorithm is inherently random, and, in general there is no way to predict—much less affect—what it will produce in any given run.
3. The motion generated by a single controller is relatively simple. The only way to get complex, composite motion is to concatenate several problem instances in time and to generate separate motion controllers serially for each subproblem—an approach similar to one proposed originally by Cohen [2].

In this paper we show how each of these problems can be addressed. In particular, we present a new search method that runs substantially faster than the original algorithm. We also present two techniques that afford the user limited control over the search

[1] Because this optimization problem is NP-hard, there exists no polynomial-time algorithm that is guaranteed to return optimal solutions unless P=NP.

[2] The most obvious shortcoming, which is not addressed here, is that the method works only for 2D articulated figures. However, it has been shown recently that the basic approach can be extended to handle 3D articulated figures and 3D mass-spring models, albeit at significantly greater computational cost [12].

algorithm. An animator can use these techniques to influence directly the style of the computed motion. They can also be used to produce a suite of qualitatively different motion controllers for a given SC problem. Such a selection of different motion controllers can be passed as input to an animation editor that allows an animator to concatenate controllers interactively to produce composite motions. We describe such an editor, and show sample figure trajectories that were created using the techniques described in the paper.

2 The Original Approach

The system described by Ngo and Marks [10, 9] searches in the space of BSR controllers using a massively parallel GA that runs on a 4096-processor CM-2. Although we assume general familiarity with the approach as described in previous work [10, 9], we now briefly review the details of the BSR controller and the GA.

A BSR controller governs a vector $\vec{\theta}(t)$ of joint angles, given information about the physical environment in the form of a vector $\vec{S}(t)$ of *sense variables*. Sample sense variables for an articulated figure are listed in Table 1.

$\theta_1, \theta_2, \ldots, \theta_{n-1}$	Joint angles
$f_1, f_2, \ldots, f_{n+1}$	Contact forces at rod endpoints
y_{cm}	Height of center of mass
$\dot{y}_{\text{cm}}$	Vertical velocity of center of mass

Table 1: Components of the vector $\vec{S}$ of sense variables for an n-rod articulated figure.

The controller contains N stimulus-response rules. Every rule i is specified by stimulus parameters $\vec{S}^{\text{lo}}[i]$ and $\vec{S}^{\text{hi}}[i]$, and response parameters $\vec{\theta}^{0}[i]$ and $\tau[i]$. Based on the instantaneous value of the sense vector $\vec{S}(t)$, exactly one rule is active at any one time. In particular, each rule i receives a score based on how far the instantaneous sense vector $\vec{S}(t)$ falls within the hyperrectangle whose corners are $\vec{S}^{\text{lo}}[i]$ and $\vec{S}^{\text{hi}}[i]$. The highest-scoring rule is said to be marked *active*. (If $\vec{S}(t)$ is not inside the hyperrectangle associated with any rule, the rule active in the previous time step remains active.) The joint angles $\vec{\theta}(t)$ are made to approach the target values $\vec{\theta}^{0}[i_{\text{active}}]$ prescribed by the active rule i_{active}.

The following pseudocode summarizes how a BSR controller behaves and is evaluated:

```
Set i_active to 1
for t = 1 to t_max
  Cause joint angles θ(t) to approach θ^0[i_active]
    with time constant τ[i_active]
  Simulate motion for time interval t
  Measure sense variables S(t)
  Possibly change i_active, based on S(t)
end for
Assign the controller a fitness value based on
  how well the simulated motion meets the
  animator-supplied task criteria
```

The original search algorithm used by Ngo and Marks was a massively parallel genetic algorithm (GA). In a GA, a population

```
do parallel
  Randomize genome
end do
for generation = 1 to number_of_generations
  do parallel
    Evaluate genome
    Select mate genome from a nearby processor
    Cross genome with mate genome
    Mutate new genome
  end do
end for
```

Figure 1: A parallel GA.

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, combining with another genome, and dying, based on its fitness value. In their generational-replacement GA, presented in Figure 1, each processor was responsible for evaluating one genome per generation in parallel.

The most important factor in the success of the GA is the approach of searching in the space of possible motion controllers, rather than the space of trajectories. Nevertheless, certain secondary details of the initial random-generation process, the mate-selection process, and the crossover and mutate operators can be important for successful motion synthesis and are described elsewhere [10, 9]. Except where noted, we have used the same process for initial random generation and the same crossover and mutate operators in the algorithms described here.

3 Accelerating the Search

The match between SIMD massive parallelism and the generational-replacement GA used by Ngo and Marks seems at first to be ideal:

- With one candidate solution per processor, and comparable processing required for each solution, it appears possible to keep all the processors busy most of the time.
- Only local communication between processors is necessary, obviating the need for expensive global communication in the processor network.

However, the issue of suitability is more complex [8]:

- SIMD parallelism is best exploited if the population size is at least as large as the number of processors in the machine—4096 on the CM-2 used by Ngo and Marks [10]. However, 4096 is probably too large for this application if the object is to find solutions of high fitness using the minimum number of evaluations.
- A SIMD architecture usually rules out the use of a steady-state GA, which tends to require fewer evaluations than a generational-replacement GA to achieve similar levels of fitness [4].

```
Initialize population
for generation = 1 to number_of_generations
  Evaluate each genome in population
  for i = 1 to (size_of_population / 2)
    Select two parent genomes by roulette-wheel selection
    Cross & mutate to generate two child genomes
  end for
  Replace old parent population with new child population
end for
```

Figure 2: A generational-replacement GA (GGA).

```
Initialize population
Evaluate each genome in population
Rank order the population
for evaluation = 1 to (number_of_evaluations / 2)
  Select two parent genomes by rank-based selection
  Cross & mutate to generate two child genomes
  Evaluate the two child genomes
  Insert child genomes in order into population
  Delete two lowest-ranked genomes in the population
end for
```

Figure 3: A steady-state GA (SSGA).

- When nonholonomic constraints (such as those imposed by the ground) are present, a physical simulator can contain a high proportion of conditionally executed code, which translates into idle time on a SIMD machine.

To assess properly the expected computational cost of synthesizing BSR controllers, and to try to discover a better search algorithm that would run on a conventional workstation, we embarked on an experimental analysis[3] of many different stochastic search algorithms—including both GA and non-GA strategies—for a selection of SC problems. A representative set of algorithms is described in Figures 2-5.

The salient feature of the generational-replacement GA (Figure 2) is that the whole genome population gets replaced every iteration. Parent genomes are selected with a bias toward fitter individuals. (In particular, the GGA described here employs a technique called roulette-wheel selection [6].) Two child genomes are produced from the parents by crossover: one child is a copy of one parent, and the other is a true hybrid. The child genomes next undergo mutation. When the population of child genomes has been generated, it replaces the population of parent genomes completely. Of the serial algorithms that we tested, this one is the most like the Ngo-Marks massively parallel GA (Figure 1), though it contains no analogue to the localized mating scheme found in the parallel algorithm.

Unlike the GGA, the steady-state GA (Figure 3) replaces at most two genomes during each iteration. Two parent genomes are selected with a bias toward fitter individuals. (The SSGA described here employs a technique called linear rank-based selection [4].) The resulting child genomes are computed just as in the GGA. They are then inserted into the population, displacing the two lowest-ranked genomes.

[3]This empirical study is described in greater detail elsewhere [5].

```
Initialize and evaluate a single genome
for evaluation = 1 to number_of_evaluations
  Randomly perturb the genome
  Evaluate the new genome
  if the new genome is better than the old one then
    Replace the old genome with the new one
  end if
end for
```

Figure 4: Stochastic hill climbing (SHC).

```
Initialize population
Evaluate each genome in population
for generation = 1 to number_of_generations
  for each individual genome in the population
    Randomly perturb the genome
    Evaluate the new genome
    if the new genome is better than the old one then
      Replace the old genome with the new one
  end for
  if (generation mod reseed_interval) = 0 then
    Rank order the population
    Replace bottom 50% of the population with top 50%
  end if
end for
```

Figure 5: Stochastic population hill climbing (SPHC).

The remaining two stochastic algorithms do not employ crossover (though we still use the terms "genome" and "population" for continuity). Stochastic hill climbing (Figure 4) is a variation on the standard hill-climbing search heuristic in which a single initial solution is perturbed randomly at each iteration, and only changes that improve the fitness score are accepted. A deficiency of the simple SHC algorithm is that the search can be trapped in a local minimum. Having a population of solutions, as in stochastic population hill climbing (Figure 5), adds robustness because of the use of multiple starting points. However, merely using a population in a simulated parallel search is equivalent to running a series of shorter, single-solution SHC trials. What distinguishes SPHC is the addition of a reseeding operation that provides a mechanism for a periodic refocusing of the search on the most promising 50% of the genomes in the population.

We tested each of these algorithms on five different SC problems that have been described previously [10, 9]. As a benchmark, we also implemented a simple random generate-and-test (RG&T) algorithm. Each algorithm was allowed to perform 40,000 evaluations (physical simulations), and the fitness score of the best result found was recorded.

No empirical comparison of stochastic algorithms is ever complete. Nevertheless, representative results are shown in Figure 6 for the problem of causing Mr. Star-Man, a five-rod articulated figure, to travel horizontally (the fitness function is the horizontal distance

Algorithm	*Population*	*Mutation Rate*	*Crossover Rate*	*Reseed Interval*
RG&T	N/A	N/A	N/A	N/A
GGA	200	0.1	0.6	N/A
SSGA	200	0.1	0.6	N/A
SHC	1	1.0	N/A	N/A
SPHC	10	1.0	N/A	200

Table 2: Key parameters.

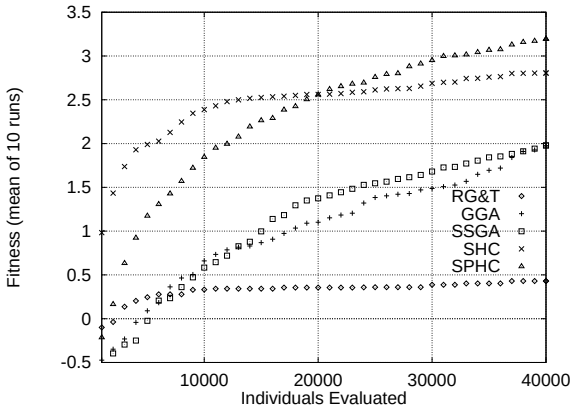

Figure 6: Comparative performance of the algorithms.

moved by the center of mass).[4] The key parameters used for the various algorithms are summarized in Table 2. The crossover and mutation rates indicate the probabilities with which a component of the genome will be subject to crossover or mutation each time a child is generated [10, 9]. Many other combinations of parameter values were tested, but for each algorithm listed, no better set of parameters was found for this problem [5]. In addition, various enhancement to the GAs (such as niching) were tested but were not found to yield significant improvement [5].

Figure 6 demonstrates the following "pecking order" among the search algorithms tested:

$$\text{SHC}, \text{SPHC} \gg \text{GGA}, \text{SSGA} \gg \text{RG\&T}.$$

These qualitative results also held true for the other four test problems.[5] The success of simple search algorithms such as SHC and SPHC reinforces the point that the space of controllers is far easier to search than the space of trajectories.

Why did the hill-climbing approaches outperform the genetic algorithms in our tests? This phenomenon, which contradicts the early expectations of the GA community, has been observed in studies with other optimization problems, and understanding when it is likely to occur is currently an active area of inquiry [3, 7].

Our study produced two other relevant results. Due to the enormous computational cost that would have been incurred on a CM-2, we did not obtain performance data for the Ngo-Marks massively parallel GA that could be compared quantitatively with the results given here. Nonetheless, the following clear anecdotal evidence sheds some light on the relative performance of the parallel algorithm:

- The parallel GA requires between 200,000 and 850,000 physical simulations to produce motion controllers comparable to those produced at a cost of 40,000 simulations by the SPHC algorithm.
- The running time of the massively parallel GA is typically 30–60 minutes on a 4096-processor CM-2 Connection Machine, versus 3–6 minutes on a DEC 3000/400 AXP workstation.

The original Ngo-Marks algorithm therefore appears to be extremely inefficient relative to the SPHC algorithm.

The other relevant result concerns the application of standard function-optimization algorithms to the problem of finding good BSR controllers for SC problems. We investigated the use of two techniques, the Downhill Simplex Method of Nelder and Mead, and Powell's Method [13]. We found that these deterministic algorithms did not generate good BSR controllers from random starting points. However, in many cases they found small to medium improvements in solutions originally generated by the SPHC algorithm.

4 Influencing the Search

The second problem we address in this paper is the inability of the user to influence the search algorithm in any way. This causes difficulties when the animator has a preconceived idea for how an animated character should move, but is unable to cause the search algorithm to generate the expected motion. It is also a problem when what is required is not just a controller for one kind of motion, but a suite of controllers for many different kinds of motion for the same animated character (§5).

4.1 Additional fitness terms

A concrete instance of this problem is provided by Mr. Star-Man. Given the task of making Mr. Star-Man travel, the search algorithms usually find a motion controller that produces a shuffling gait like the one in Figure 8. Occasionally, when monitoring the progress of the search, we noticed motion controllers that produced a cartwheel (Figure 9), but it was rare for the search to converge on such a controller. We therefore set out to provide some mechanism whereby we could reliably guide the search toward either the shuffle or the cartwheel.

One useful mechanism we discovered was to use *secondary terms* in the fitness function. The *primary term* in the fitness function is the one given the most weight, and is therefore the one that determines the most salient characteristic of the motion. For example, to get Mr. Star-Man to travel, *i.e.*, to move from left to right, the primary term in the fitness function is the horizontal distance traversed by his center of mass. The secondary terms, which are simply added to the primary term in the fitness function, determine minor characteristics of the motion. The secondary terms we investigated are given in Table 3. For the sample problem we proposed above, we found that assigning a positive weight only to the

[4] Two typical good solutions to this problem are shown in Figures 8 and 9.

[5] One of the top performers, SHC, is very similar in form to the stochastic gradient ascent used by Van de Panne and Fiume [16] to refine SANs. However, in our tests, the starting point for SHC was always a solution drawn from a random initial distribution, not one produced by an earlier stage in the search process.

Term	*Function*
`max_cm_height`	Maximum height of center of mass during motion
`slip_sum`	Distance traversed by body parts in continuous contact with ground
`rotations`	Number of full-body rotations during motion

Table 3: Secondary terms in the fitness function.

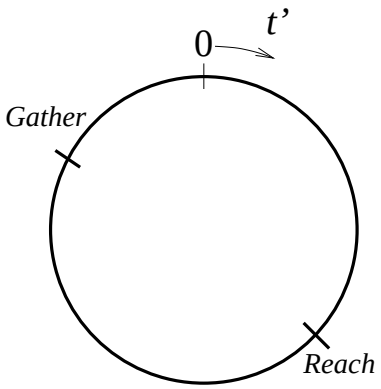

Figure 7: Simplified diagram of a BSR controller in which t' is the sole sense variable.

secondary term `slip_sum` guarantees a shuffling gait, whereas assigning a positive weight only to the secondary term `rotations` guarantees a cartwheel. For our other animated characters, we found that varying the coefficients of the small set of terms in Table 3 was sufficient to generate reliably all motions that we had ever seen occur at random, plus others. Moreover, the motions generated correlate qualitatively with the secondary fitness terms: a positive coefficient for `max_cm_height` biases the search in favor of hopping or jumping motions, a positive coefficient for `slip_sum` encourages sliding or shuffling, and a positive coefficient for `rotations` usually leads to some kind of gyration.

4.2 Different sense variables

The nature of the motions produced by controller synthesis depends intimately on the space of controllers made available for searching. (For example, BSR controllers produce motions that are quite different from those produce by SANs [11, 17].) Thus, the second way we propose to influence the style of generated motions is to modify the form of the BSR controller.

We modified the BSR controller by substituting a periodically resetting timer value $t' \equiv t \bmod t_p$ for the entire set of physical sense variables listed in Table 1. (The period t_p is a constant chosen by the animator.) The effect is to reduce the BSR controller to a mechanism of the type found in a child's "wind-up" toy—a device that executes a sequence of motions repetitively, independent of the state of the physical environment (Figure 7).

In recent papers [10, 9, 16] it has been argued or assumed that the space of controllers with access to information about physical state should be easier to search than the space of controllers based on time alone. Unexpectedly, we found that good motions are easier to compute when t' is the only sense variable than when physical sense variables (in our tests, those listed in Table 1) are used.[6] Furthermore, by manually varying the value of the period t_p, it is possible to elicit a variety of motions, from highly periodic to completely aperiodic. For example, the "tumbling" sequence of Beryl Biped depicted in Figure 10 was computed with $t_p > T$, where T is the length of time available for the entire motion. The periodic shuffling in Figure 11 was computed with a value of $t_p \ll T$.

5 The Animation Editor

The automatic motion-synthesis techniques described here and elsewhere are currently capable of producing controllers for relatively simple motions only. To produce more complex movement, Cohen [2] implemented a system that permits a human animator to design and refine a trajectory by interactively submitting simple subproblems for solution on-line. This approach may be quite attractive for 2D articulated figures, especially given the ability to solve 2D SC problems in times ranging from seconds to minutes on a modern workstation (§4.2). However, such speedy turnaround is not likely for 3D SC problems in the near future [12].

As an alternative to Cohen's approach, we propose the following scenario for creating complex, composite motion sequences:

1. The animator defines the animated character by specifying the physical structure of a 2D articulated figure.
2. The computer generates a suite of different motion controllers for this character automatically off-line, using the SPHC algorithm (§3) and various combinations of primary and secondary fitness terms (§4.1).
3. The animator develops composite motions by interactively concatenating selected controllers in time using a graphical animation editor.

This approach has the advantage that all lengthy computations are performed off-line in step 2; the animator's interaction with the editor in step 3 does not require the solution of additional SC problems. However, it is not immediately obvious how the concatenation operation in step 3 can be supported: if a motion controller is invoked for an articulated figure that starts out in a different configuration than the one for which the controller was constructed, what happens? Fortunately, we discovered that motion controllers which use standard SR rules (§1) have a degree of intrinsic robustness. If the figure is in a configuration that is reasonably close to some configuration that occurs somewhere in the normal trajectory (the one followed by the figure when started from its expected initial configuration), then it will usually produce a motion similar to the one it was designed to produce. Thus the animator need not be intolerably precise in choosing when to switch from one motion controller to another in order to get a desired composite motion.

An example of a composite motion produced with the help of our animation editor is shown in Figure 12. Three separate motion

[6] Good time-based controllers for these problems can be found in a few tens of seconds on a DEC 3000/400 AXP workstation. One contributing factor to the ease with which good time-based controllers can be found may be the smaller number of parameters per rule.

controllers are concatenated (via a direct-manipulation GUI) to produce first a left-to-right shuffle (A–C repeated), then a right-to-left cartwheel (C–J), then an unusual jump (J–M),[7] and finally the shuffle again (M–O repeated).

6 Conclusions and Further Work

In this paper we have described an empirical study of search algorithms for 2D articulated-figure SC problems, mechanisms for influencing the various search algorithms, and an animation editor that supports the temporal concatenation of existing motion controllers. We anticipate at least three directions for future research.

First, further exploration of mechanisms for influencing the search is certainly appropriate. It is clear that such mechanisms could take many forms. Here is a partial listing of the possibilities:

- *Human as fitness function ("interactive evolution").* No quantitative objective function is used. Instead, the human animator scores every motion generated by the computer. This possibility is analogous to previous approaches to the evolution of static images [14, 15]. In a variant of this approach, controllers produced by the usual batch procedure could be refined by interactive evolution with the goal of introducing desirable nuances or subtlety of movement into a figure's trajectory.
- *Direct human control of fitness function*. As in the approach that we have employed here, the human modifies terms in the fitness function to elicit the desired motions.
- *Indirect human control of fitness function by example*. In this approach, the human animator scores some small fraction of the generated motions, and the machine tries to infer an appropriate fitness function to fit the given choices.
- *Human augmentation of search process.* In this approach, the fitness function is fixed, and the role of the human being is to encourage particular motions via gestural hints [18].

Second, to enhance the practicality of the animation editor, we are investigating how to bias the search algorithm in favor of motion controllers that function well from a large number of starting conditions.

Finally, all of these results need to be verified, and perhaps adapted somewhat, (a) for physical simulators that are more complex and realistic than the one employed here, and (b) for other autonomous animated characters, *e.g.*, mass-spring models and 3D articulated figures [12].

7 Acknowledgments

We thank Stuart Shieber for comments on the manuscript. 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, and by an equipment grant from Digital Equipment Corp. The animation-editor project was investigated on a Hewlett-Packard workstation donated by Walter Hewlett to Dunster House, Harvard University.

[7] Mr. Star-Man jumps by rapidly bringing his legs together, which elevates his center of mass rapidly. Given that we have not limited his thigh strength, this is a reasonable method of jumping, though it is not particularly anthropomorphic.

References

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

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

[3] L. Davis. Bit-climbing, representational bias, and test suite design. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 18–23, San Mateo, CA, 1989. Morgan Kaufmann.

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

[5] A. Fukunaga, J. T. Ngo, and J. Marks. Automatic control of physically realistic animated figures using evolutionary programming. In *Proceedings of the Third Annual Conference on Evolutionary Programming (EP94)*, San Diego, CA, February 1994. To appear.

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

[7] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, San Mateo, CA, 1993.

[8] J. T. Ngo and J. Marks. Massively parallel genetic algorithm for physically correct articulated figure locomotion. Working Notes for the AAAI Spring Symposium on Innovative Applications of Massive Parallelism, Stanford University, March 1993.

[9] J. T. Ngo and J. Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993. In press.

[10] J. T. Ngo and J. Marks. Spacetime constraints revisited. In *SIGGRAPH '93 Conference Proceedings*, pages 343–350. ACM SIGGRAPH, Anaheim, CA, August 1993.

[11] J. T. Ngo and J. Marks. Spacetime constraints revisited. ACM SIGGRAPH Video Review, Issue 96: Video Supplement to the SIGGRAPH '93 Conference Proceedings, 1993.

[12] H. Partovi, J. Christensen, A. Khosrowshahi, J. Marks, and J. T. Ngo. Motion synthesis for 3D articulated figures and mass-spring models. In review, 1994.

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK, second edition, 1992.

[14] K. Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328, July 1991.

[15] S. Todd and W. Latham. *Evolutionary Art and Computers*. Academic Press, London, 1992.

[16] M. van de Panne and E. Fiume. Sensor-actuator networks. In *SIGGRAPH '93 Conference Proceedings*, pages 335–342, Anaheim, CA, August 1993. ACM SIGGRAPH.

[17] M. van de Panne and E. Fiume. Sensor-actuator networks. ACM SIGGRAPH Video Review, Issue 96: Video Supplement to the SIGGRAPH '93 Conference Proceedings, 1993.

[18] J. Ventrella. Personal communication.

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

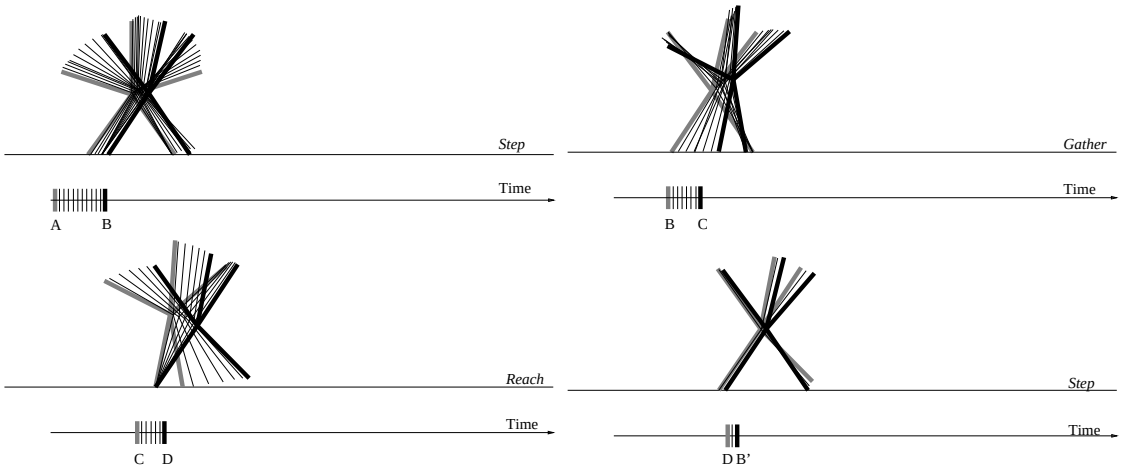

Figure 8: Mr. Star-Man shuffling. The B–C–D–B' sequence is repeated cyclically.

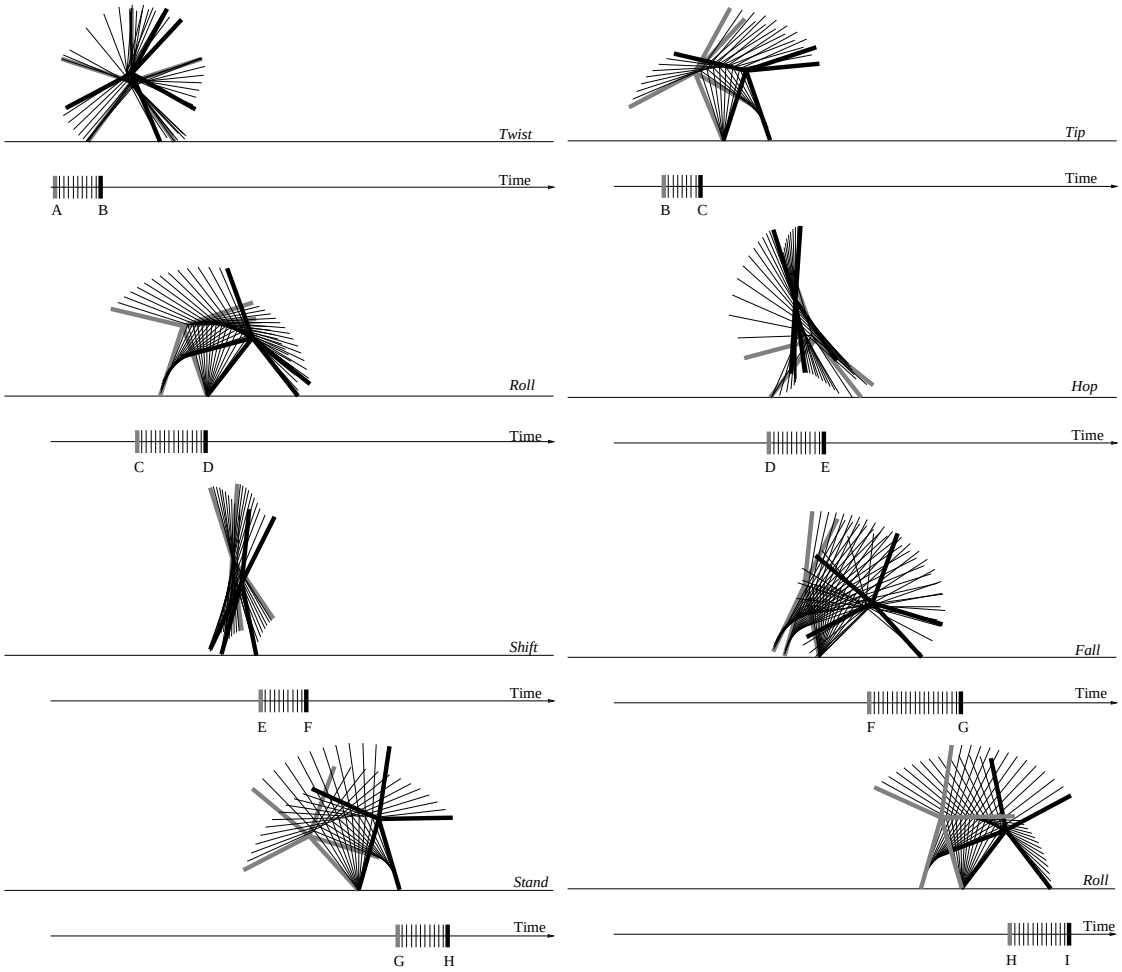

Figure 9: Mr. Star-Man doing a cartwheel.

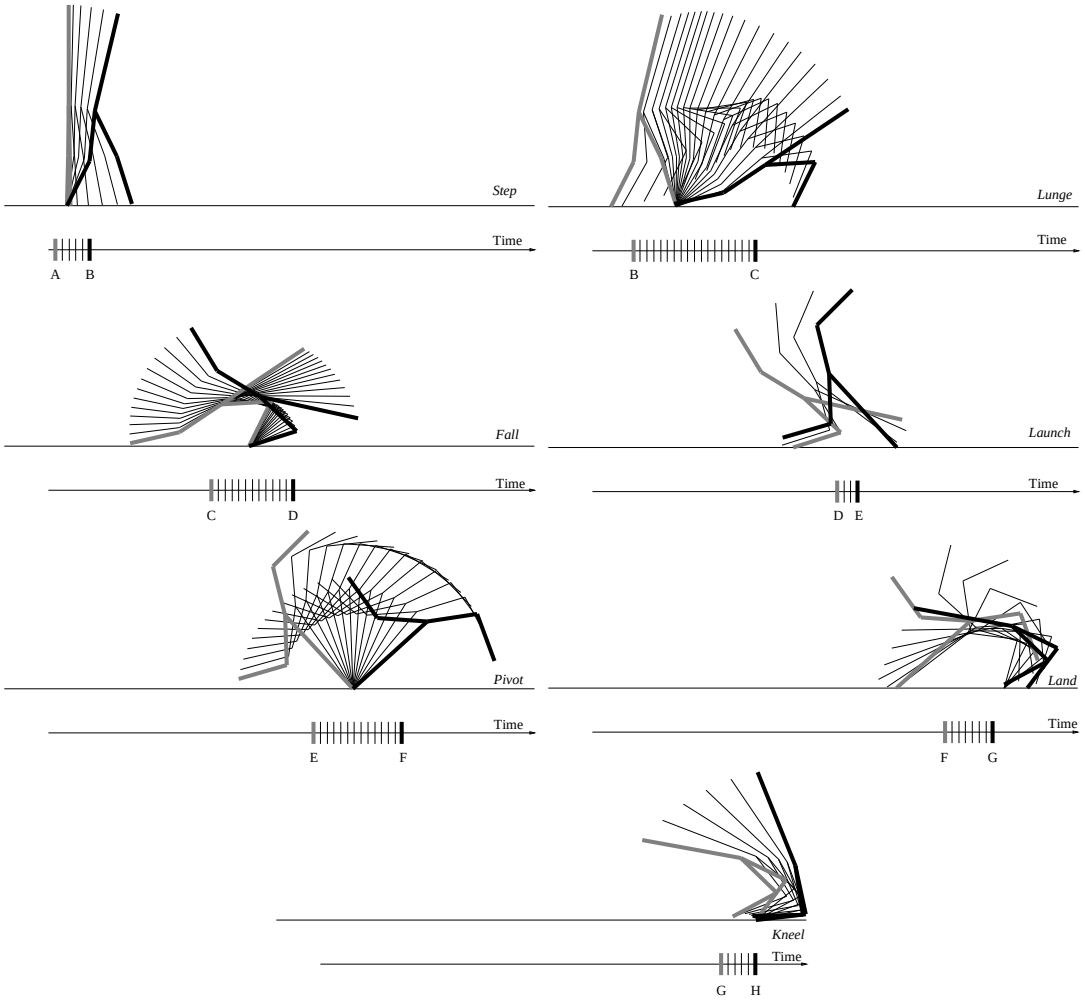

Figure 10: Beryl Biped tumbles aperiodically.

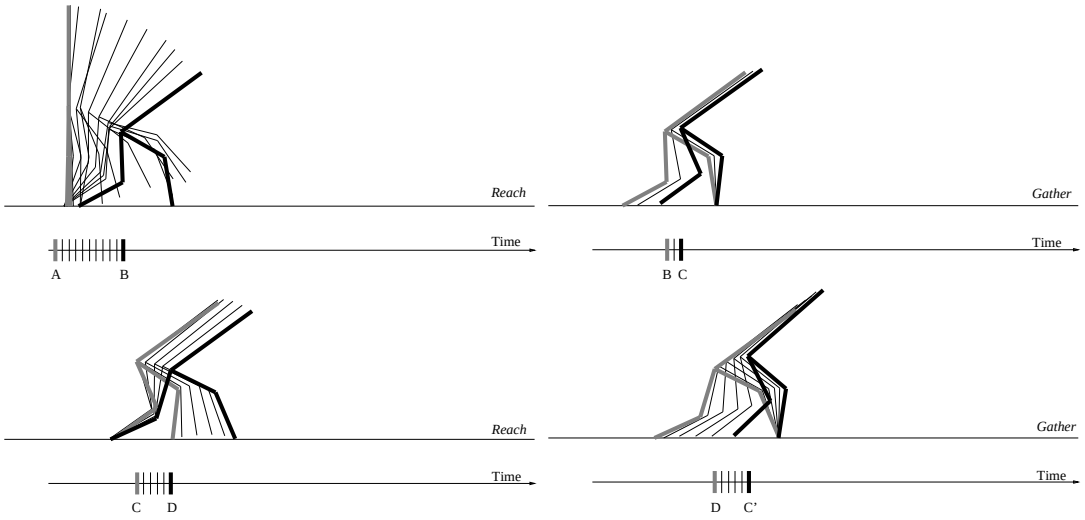

Figure 11: Beryl Biped shuffles periodically. The C–D–C' sequence is repeated cyclically.

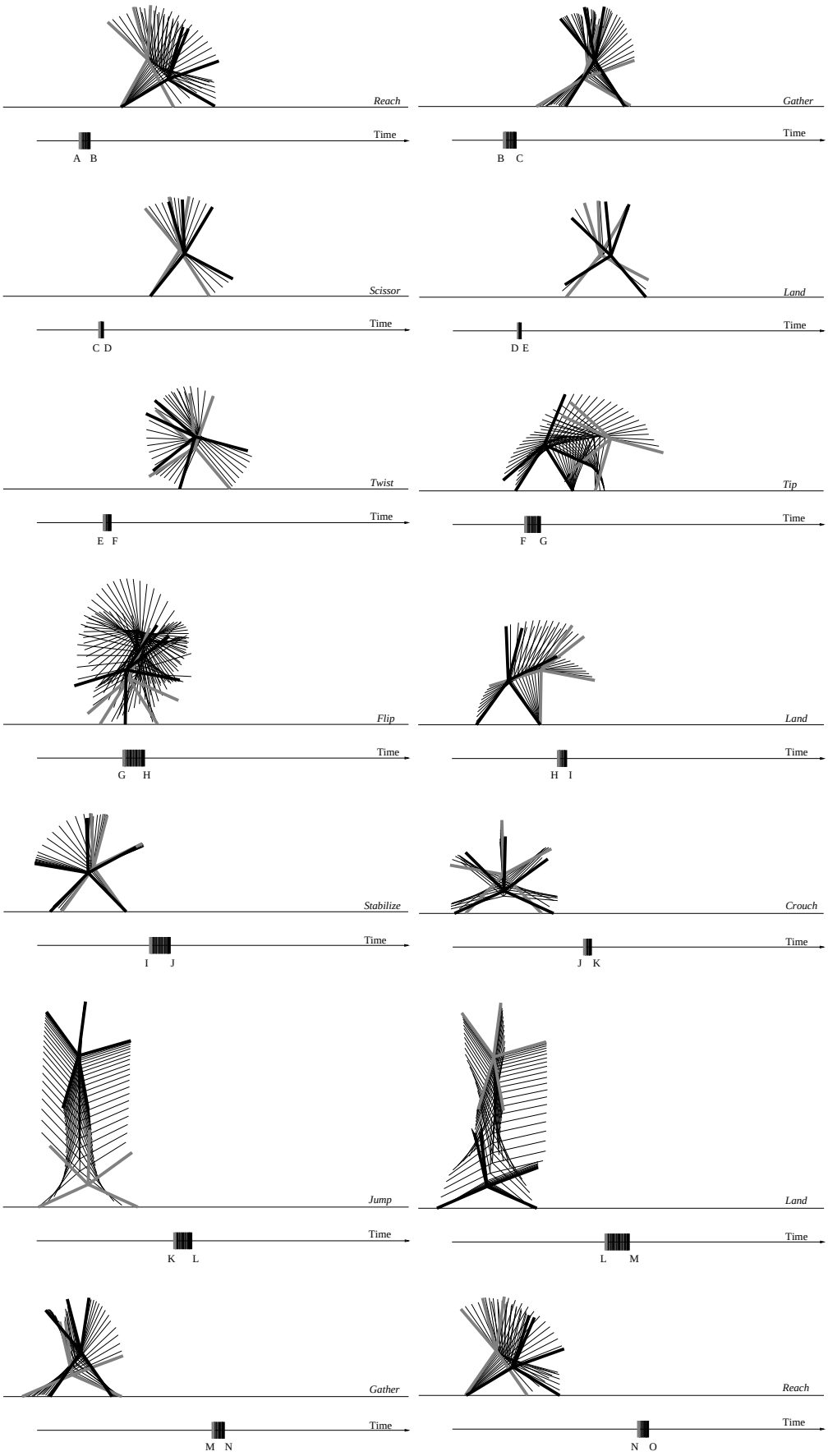

Figure 12: Mr. Star-Man's composite trajectory.