AUTOMATIC CONTROL OF PHYSICALLY REALISTIC ANIMATED FIGURES USING EVOLUTIONARY PROGRAMMING

ALEX FUKUNAGA*
Division of Applied Sciences, Harvard University
Cambridge, MA 02138, U.S.A.

and

JOE MARKS†
Cambridge Research Lab, Digital Equipment Corporation
Cambridge, MA 02139, U.S.A.

and

J. THOMAS NGO‡
Graduate Biophysics Program, Harvard University
Cambridge, MA 02138, U.S.A.

ABSTRACT

The ideal system for computer animation would relieve the human animator of responsibility for specifying details of motion. In this paper, we describe an empirical study of evolutionary algorithms for this, the motion-synthesis problem. It is shown that simple search algorithms based on the evolutionary-programming paradigm were most efficient in searching the space of candidate motion controllers for a variety of motion-synthesis problems involving 2D articulated figures.

1. Introduction

Computer animation has traditionally been a tedious and time-consuming process. In most current computer-animation systems, the animator is given minimal assistance by the computer. Automatic motion synthesis—whereby the animator specifies only the physical structure of a character and criteria for evaluating the character's motion, and the computer generates a near-optimal, physically realistic (and therefore visually plausible) trajectory for the character[7]—is much more attractive, though very difficult to realize.

To date, the most promising approach to the motion-synthesis problem has been the automatic generation of *motion controllers* that, when executed in a simulated physical environment, produce a desired motion.[5,6] (The motion-synthesis problems considered here and in the cited articles involve characters modeled as simple 2D

*Current affiliation: University of California, Los Angeles.

†Current affiliation and contact: Mitsubishi Electric Research Laboratories, Inc., 201 Broadway, Cambridge, MA 02139, U.S.A. E-mail: marks@merl.com.

‡Current affiliation: Interval Research Corp., Palo Alto.

articulated figures.) Incarnations of this approach differ in how the motion controller is represented and how the space of possible controllers is searched. Ngo and Marks[5,4] proposed an algorithm in which the controller is a bank of independent stimulus-response (SR) rules; the controllers are referred to as banked stimulus-response (BSR) controllers. Information from the physical environment determines which SR rule is active at any given time step in the physical simulation. The space of possible controllers is searched using a massively parallel GA. This approach produces very effective controllers, but the cost of the search algorithm is high: 30-60 minutes on a 4096-processor CM-2 Connection Machine to find a motion controller for a simple articulated-figure character.

The goal of our research was to develop an improved search algorithm. In this paper, we present several alternative evolutionary-computation (EC) algorithms for finding good BSR controllers and compare them empirically.

2. The Old Approach

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.

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

$\theta_1, \theta_2, \ldots, \theta_{n-1}$	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

The controller contains N stimulus-response rules ($N = 10$ for the empirical tests reported here). 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 marked *active*. If $\vec{S}(t)$ is not inside the hyperrectangle associated with any rule, the rule that is 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 pseudocode in Figure 1 summarizes how a BSR controller behaves and is evaluated. The function used to measure the fitness of a controller is specific to each motion-synthesis problem. For example, if the objective is for the character to jump as high as possible, the value of the fitness function could be the greatest height achieved by the figure's center of mass during the motion.

The original Ngo-Marks approach used a massively parallel GA to search the space of possible BSR controllers. In this algorithm, shown in Figure 2, each can-

```
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
```

Fig. 1. Pseudocode for a BSR controller.

```
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
```

Fig. 2. A parallel GA.

didate solution, or *genome*, is assigned to a single processor, and each generation of genomes is evaluated in parallel.[§]

Although it first seemed that the match between the GA and SIMD massive parallelism was ideal, Ngo and Marks later observed some incompatibilities between the CM-2 architecture and the search strategy that apparently rendered the algorithm inefficient.[3] This raised the possibility that searching for good BSR controllers could potentially be done much more easily.

3. Newly Implemented Search Algorithms

The search algorithms explored in this study are presented in Figures 3–9. Various kinds of GAs are described in Figures 3–5; Figures 6–9 contain algorithms more in the EP tradition. In addition, random generate and test (RG&T) was included as a baseline benchmark.

[§] Further algorithmic details, in particular those concerning the crossover and mutation operators, are described elsewhere.[5,4]

```
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[2]
    Cross & mutate to generate two child genomes
  end for
  Replace old parent population with new child population
end for
```

Fig. 3. 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 linear 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
```

Fig. 4. A steady-state GA (SSGA).

```
Initialize all demes
Evaluate each genome in population
Rank order each deme
for evaluation = 1 to (number_of_evaluations / (2×number_of_demes))
  for each deme
    Insert any inbound migrants into deme
    Select two parent genomes by linear 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
    With small probability
      Select two migrant genomes by linear rank-based selection
      Send copies of migrant genomes to randomly selected deme
  end for
end for
```

Fig. 5. A distributed GA (DGA).

```
Initialize population
for generation = 1 to number_of_generations
  Evaluate each genome in population
  for i = 1 to size_of_population
    Select a parent genome by roulette-wheel selection
    Mutate to generate a child genome
  end for
  Replace old parent population with new child population
end for
```

Fig. 6. Evolutionary Programming (EP1).

```
Initialize population
Evaluate each genome in population
Rank order the population
for evaluation = 1 to number_of_evaluations
  Select a parent genome by linear rank-based selection
  Mutate to generate a child genome
  Evaluate the child genome
  Insert child genome in order into population
  Delete the lowest-ranked genome in the population
end for
```

Fig. 7. Evolutionary programming (EP2).

```
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
```

Fig. 8. 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
```

Fig. 9. Stochastic population hill climbing (SPHC).

Each of these algorithms was tested on five different motion-synthesis problems, though results for only one of the five problems is reported here. (The other results are similar.) To facilitate a fair comparison, each algorithm was permitted 40,000 calls to the fitness function. Following indications from small early tests, each algorithm was tested using several sets of promising control parameters, summarized in Table 2. Some parameters in the table require a little explanation: the crossover and mutation rates give the probability that these operators are applied to a genome; the migration rate indicates the probability with which outbound migrants are spawned from a particular deme in the distributed GA; and the reseed interval is the number of generations between reseeding operations in the SPHC algorithm.

4. Results

The relative performance of the various algorithms (using the best parameters for each algorithm) for a representative motion-synthesis problem involving a character called Mr. Star-Man[5,4] is illustrated in Figure 10.¶ The measure of performance is the average fitness value from 10 runs of the best controller evaluated up to that point.

From the figures, the following general observations can be made:

- As a group, the EP algorithms outperformed everything else. In particular, the SPHC algorithm consistently yielded the best results.
- All the EC algorithms outperformed RG&T.
- There is no consistent trend among the GAs.

¶The complete set of data for this study is presented elsewhere.[1]

Table 2. Key parameters.

Experimental Group	*Population*	*Mutation Rate*	*Crossover Rate*	*Migration Rate*	*Reseed Interval*
RG&T	N/A	N/A	N/A	N/A	N/A
GGA-1	100	0.1	0.6	N/A	N/A
GGA-2	200	0.1	0.6	N/A	N/A
SSGA-1	100	0.1	0.6	N/A	N/A
SSGA-2	200	0.1	0.6	N/A	N/A
DGA-1	5 X 40	0.1	0.6	0.05	N/A
DGA-2	5 X 40	0.1	0.6	0.005	N/A
DGA-3	5 X 100	0.1	0.6	0.05	N/A
DGA-4	5 X 100	0.1	0.6	0.005	N/A
EP1-1	20	1.0	N/A	N/A	N/A
EP1-2	50	1.0	N/A	N/A	N/A
EP2-1	20	1.0	N/A	N/A	N/A
EP2-2	50	1.0	N/A	N/A	N/A
SHC	1	1.0	N/A	N/A	N/A
SPHC-1	10	1.0	N/A	N/A	100
SPHC-2	10	1.0	N/A	N/A	200
SPHC-3	10	1.0	N/A	N/A	400
SPHC-4	10	1.0	N/A	N/A	400
SPHC-5	10	1.0	N/A	N/A	800

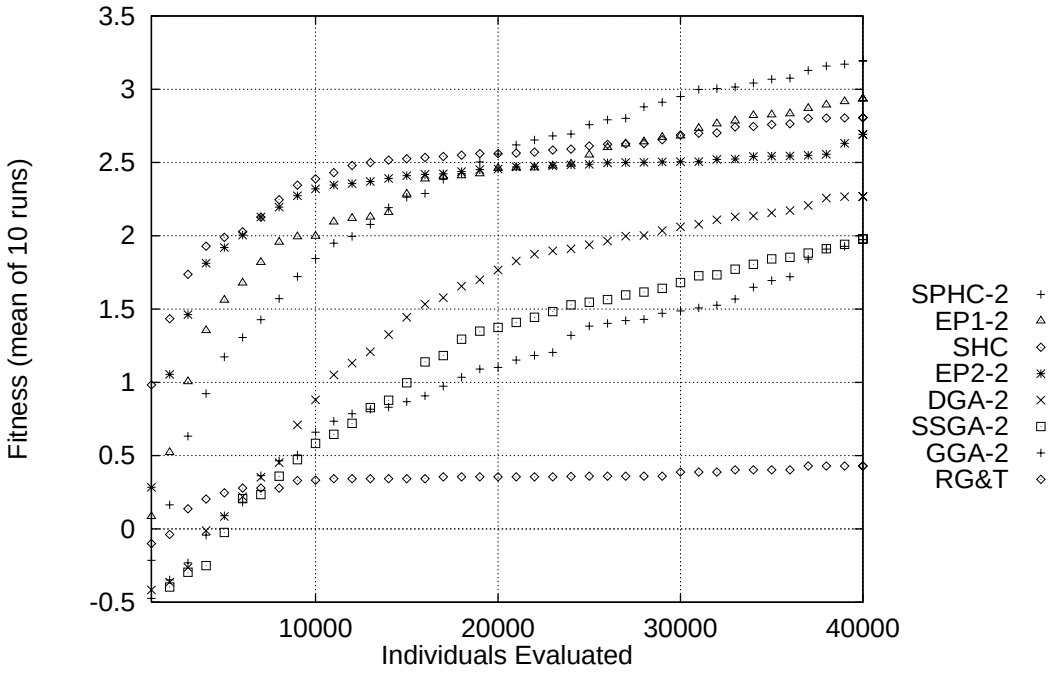

Fig. 10. Comparative performance of the search algorithms.

Due to the enormous computational cost that would have been incurred on a CM-2, we did not obtain performance data for the massively parallel GA (Figure 2) that could be compared quantitatively with the serial algorithms described here. However, we can make the following anecdotal observations about the relative performance of the serial and parallel algorithms:

- The parallel GA requires between 200,000 and 850,000 physical simulations (50-200 generations of the GA) 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 to produce comparable results.

5. Conclusions

This study has shown that evolutionary programming is particularly effective for finding near-optimal solutions to the motion-synthesis problem. The success of simple EP algorithms such as SPHC showed that the space of BSR controllers is relatively simple to search, supporting the view that generating motion controllers is an attractive approach to motion synthesis.

6. References

1. A. Fukunaga, *Genetic and stochastic search strategies to solve the spacetime constraints problem.* Undergraduate thesis, Harvard University, April 1993.
2. D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, Massachusetts, 1988).
3. J. T. Ngo and J. Marks, *Massively parallel genetic algorithm for physically correct articulated figure locomotion*, in Working Notes for the AAAI Spring Symposium on Innovative Applications of Massive Parallelism, Stanford University, March 1993.
4. J. T. Ngo and J. Marks, *Physically realistic motion synthesis in animation*, Evolutionary Computation, **1** (1993), pp. 235–268.
5. J. T. Ngo and J. Marks, *Spacetime constraints revisited*, in SIGGRAPH '93 Conference Proceedings (ACM SIGGRAPH), Anaheim, CA, August 1993, pp. 343–350.
6. M. van de Panne and E. Fiume, *Sensor-actuator networks*, in SIGGRAPH '93 Conference Proceedings (ACM SIGGRAPH), Anaheim, CA, August 1993, pp. 335–342.
7. A. Witkin and M. Kass, *Spacetime constraints*, Computer Graphics, **22** (1988), pp. 159–168.