

Evolutionary Design I

Jason Noble

`jasonn@comp.leeds.ac.uk`

Biosystems group, School of Computing

This lecture

- Harnessing evolution in a computer program
- How to construct a genetic algorithm
- Potential difficulties, alternative approaches
- Some history
- Applications: what are evolutionary algorithms good for?

Artificial evolution

Why should we be interested?

- If real evolution produced the rich variety and superb adaptation of the biosphere, then maybe we can use artificial evolution to get a piece of the action.
- In addition, maybe we can use artificial evolution to better understand how the phenomena of the biosphere came to be.

Like many of the topics discussed in this course, artificial evolution has a dual life as an engineering technique and as a scientific modelling method.

Real biology

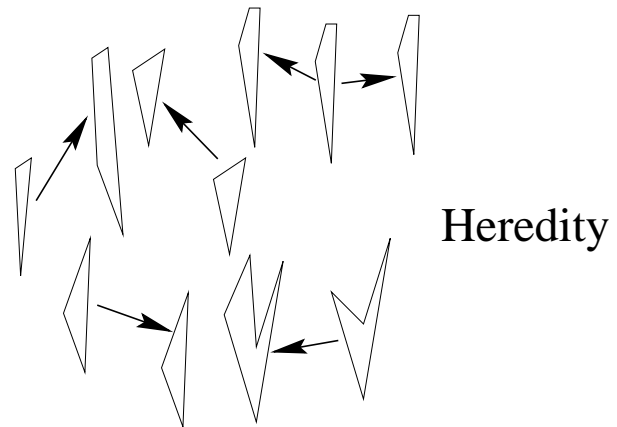
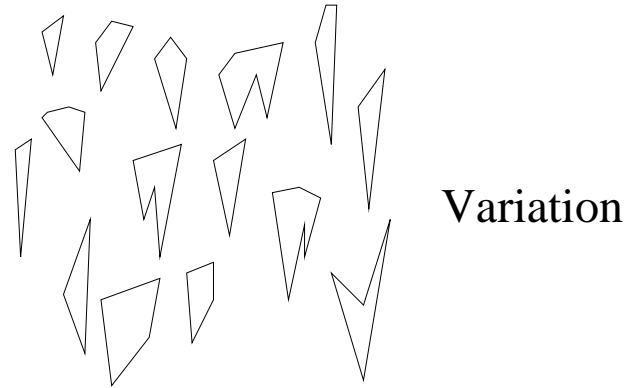
Evolution requires a population of replicating individuals with *all* of the following features (Darwin, 1859):

Variation: individuals not all the same; some random variation present.

Selection: not all individuals survive to reproduce, and some individuals reproduce more than others.

Heredity: individuals tend to be like their parents.

The bare essentials



Natural evolution

Biological organisms are *self*-replicating: impressive, but not a necessary condition for evolution.

In nature:

- Variation occurs because of DNA transcription errors.
- Selection occurs because organisms are competing for limited food, space, mating opportunities, etc.
- Heredity is observed because the DNA “recipe” is passed on.

But nothing in the theory limits its application to terrestrial biology. Note that Darwin was ignorant of the *mechanisms* of variation and heredity.

Evolution as an algorithm

It is possible to implement variation, selection and heredity in a computer program.

To evolve solutions to a problem we need:

- A way of representing candidate solutions; something analogous to DNA.
- A way of calculating how good a particular solution is: a “fitness function”. High scorers on the fitness function will be selected (or low scorers if it’s an error function).
- Something analogous to reproduction, in which fit solutions are used to produce new solutions, with some degree of random variation.

Evolving pizza

An example to illustrate how GAs work.

Cheese	Ham	Pepperoni	Chillies	Mushrooms	Base	Tomato sauce	Onion	Anchovies	Olives	
1	1	1	0	0	0	0	0	0	1	0

A pizza chef wants to find his “optimal” pizza recipe. Customers will give numerical ratings for the pizzas he cooks.

The pizzas have up to 10 ingredients: we can specify a specific pizza recipe with a string of 10 binary digits. How should he go about finding his customers’ favourite pizza? Cook all $2^{10} = 1024$ possible pizzas? Cook random pizzas?

The GA strategy

1. Make a random batch of 20 pizzas (toss a coin to get the recipes).
2. Send the 20 pizzas out, and record the customer ratings.
3. To make 20 new pizzas, take the specifications for two that did well in the last batch, and cross them over at a random point to make two new specifications.
4. Occasionally make an arbitrary change to the specified recipes, e.g., leave out the onions or add mushrooms.
5. Go back to step 2. Repeat until happy.

A GA in action

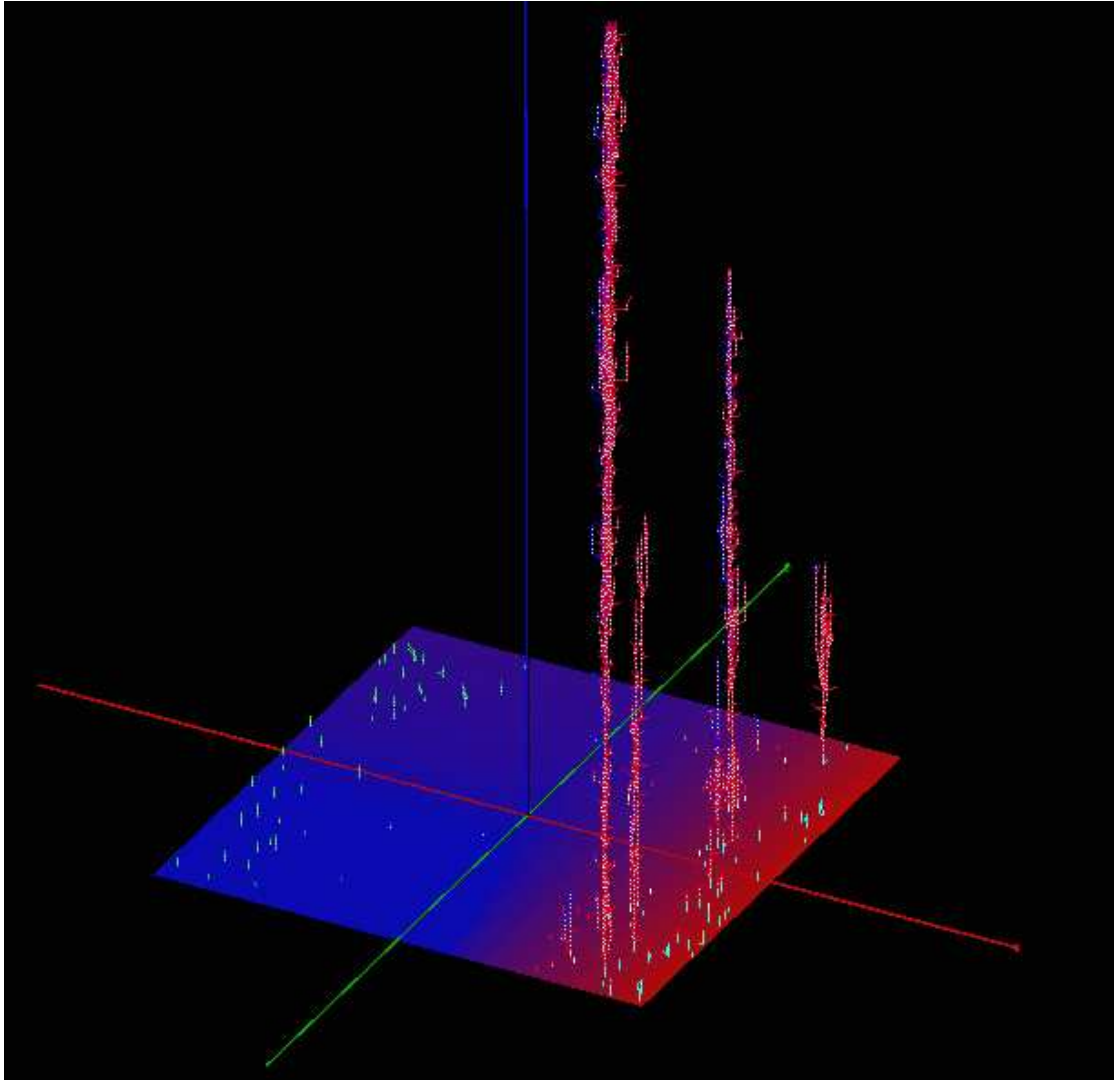


Image supplied by Martin Thompson

Components of a GA

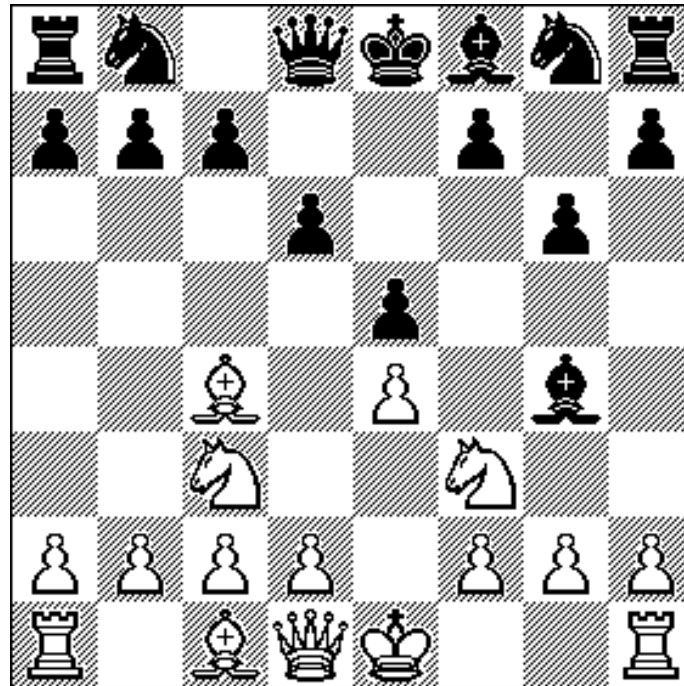
- Population of solutions / individuals
- Mapping from genotype to phenotype
- Fitness function
- Selection procedure
- Crossover
- Mutation

Population of individuals

- How big does the population need to be?
 - A trade-off.
 - Anywhere from 30 to 1000 or more.
 - Or maybe just one?
- What does an individual look like?
 - A data structure.
 - Strings of bits are popular.
 - Can be real-valued, or just about anything.

From genotype to phenotype

- Choice of mapping or *encoding scheme* is critical.
- Some problems have an obvious mapping, e.g., the pizza example.
- What if we wanted to evolve difficult chess positions?



Encoding chess positions

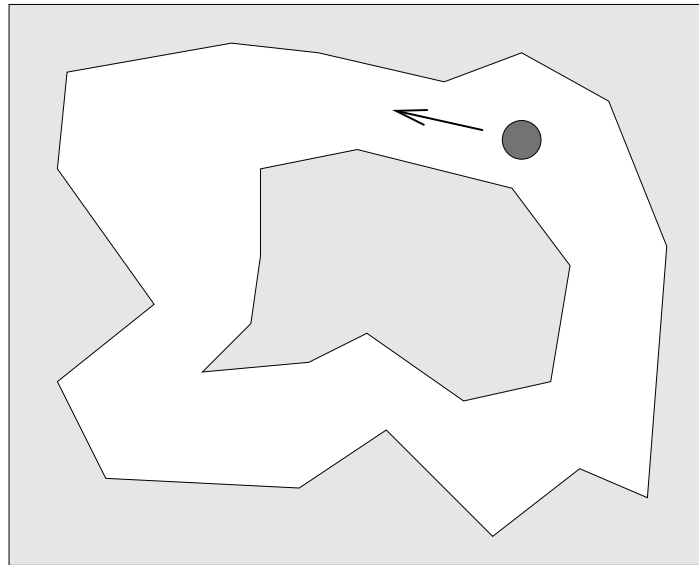
- To encode a chess position by specifying which of the 13 possible piece-types was on each square would require $64 \times \log_2 13 = 237$ bits. Specifying the location of each of the 32 pieces would take $32 \times \log_2 65 = 193$ bits.
- The second space is many times smaller, thus evolution is likely to be faster. Note: neither encoding scheme is *closed* over the space of legal positions—both allow for many impossible boards.
- Next time: size of the search space is important, but its “topography” is even more so.

Fitness functions

- Fitness functions can be hard to get right, and they may have “loopholes.”
- For example, Zaera, Cliff, and Bruten (1996) tried to evolve flocking behaviour and gave up as the fitness function proved impossible to develop.
- If you can't calculate a continuous, one-dimensional measure of solution quality, your GA is in trouble.

A fitness function example

- We want to evolve robot controllers that produce smooth movement around the track.



- Start with a fitness function that rewards collision avoidance, e.g., avg. distance from wall, over the trial. Open to “exploitation” by robots that go somewhere far from the walls and sit still.

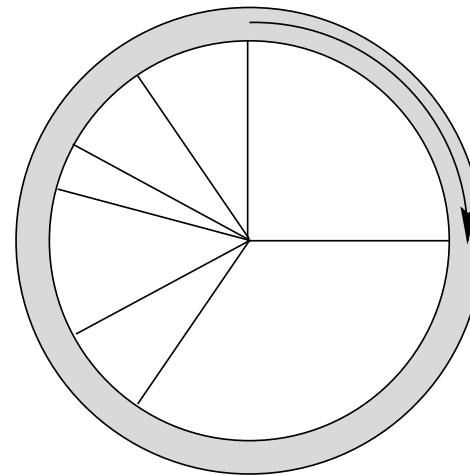
Fitness function example

- Correct by adding a second component: robots score well for keeping away from walls and for moving as far as possible. Open to exploitation by robots that sit and spin in a tight circle.
- Finally: add a requirement that penalizes tight turning angles, and you get the behaviour you wanted in the first place (Floreano & Mondada, 1994).
- When the fitness function has to be tuned so carefully, evolution looks like less of a free lunch.
- Another problem: multi-dimensional optimization. Is a weighted sum good enough? Will return to this point in co-evolution lectures.

Selection procedure

- A selection operator chooses fit solutions for reproduction.
- For example, roulette-wheel selection, where the probability of a solution being selected is equal to its fitness divided by the total fitness.

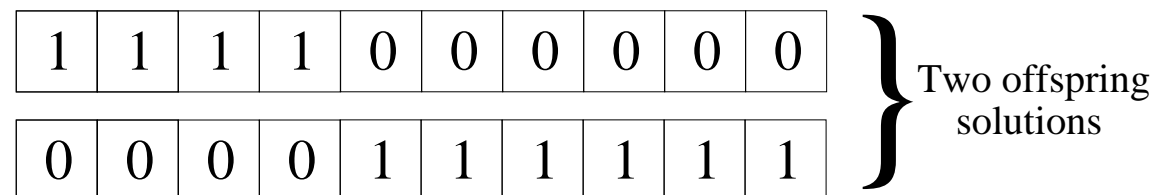
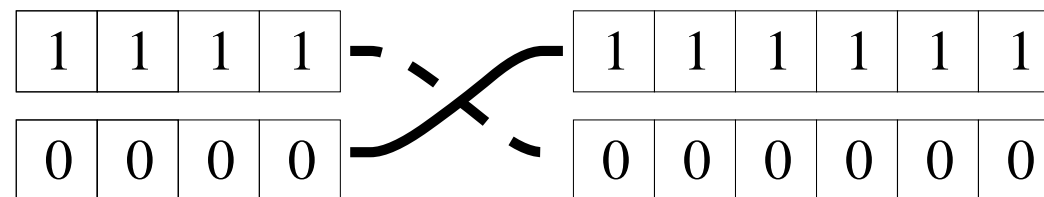
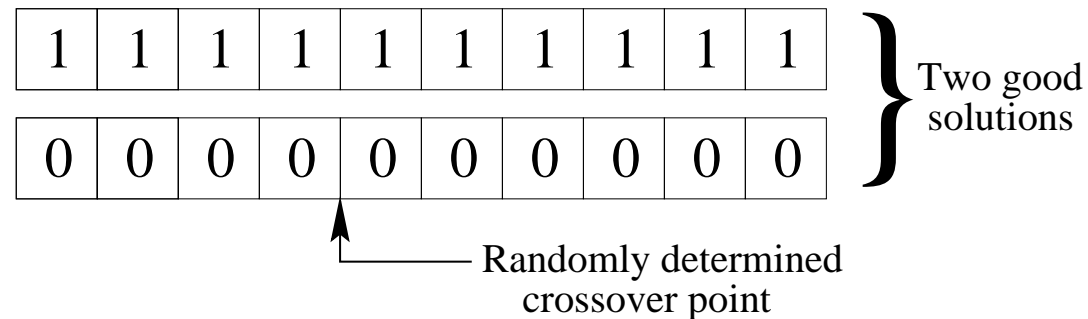
$$p_R(j) = \frac{f(j)}{\sum_{i=1}^n f(i)}$$



- Other possibilities: rank-based selection, tournament selection.

Crossover

- The idea of crossover is that advantageous mutations can be shared around: compare sexual and asexual reproduction in nature.



Crossover

- One-point crossover is the simplest form.
- Multi-point crossover, uniform crossover also possible.
- The choice of method may not be very important.

Mutation

- Mutation is the source of variation.
- Without mutation, crossover could shuffle the initial set of “genes” around, but there would be no evolutionary novelty.
- In a bit-string, mutation is implemented as a probability that any one bit will be flipped during reproduction.
- Things get trickier with real-valued genotypes, and a poorly chosen mutation operator may introduce biases (Bullock, 1999).
- Choosing the mutation probability: $1/N$? More? Less?

Cautionary notes

- Setting parameters can be a black art (e.g., mutation rate, population size).
- Genotype–phenotype mapping makes a difference.
- Fitness function can be so time-consuming to develop that it negates the point of the exercise.

A bit of history

In the 50s, 60s and 70s, various people tried designing evolutionary algorithms.

- Holland (1975). Genetic algorithms (GAs): candidate solutions encoded as fixed-length binary strings.
- Rechenberg and Schwefel. Evolution strategies: candidate solutions encoded as lists of real numbers.

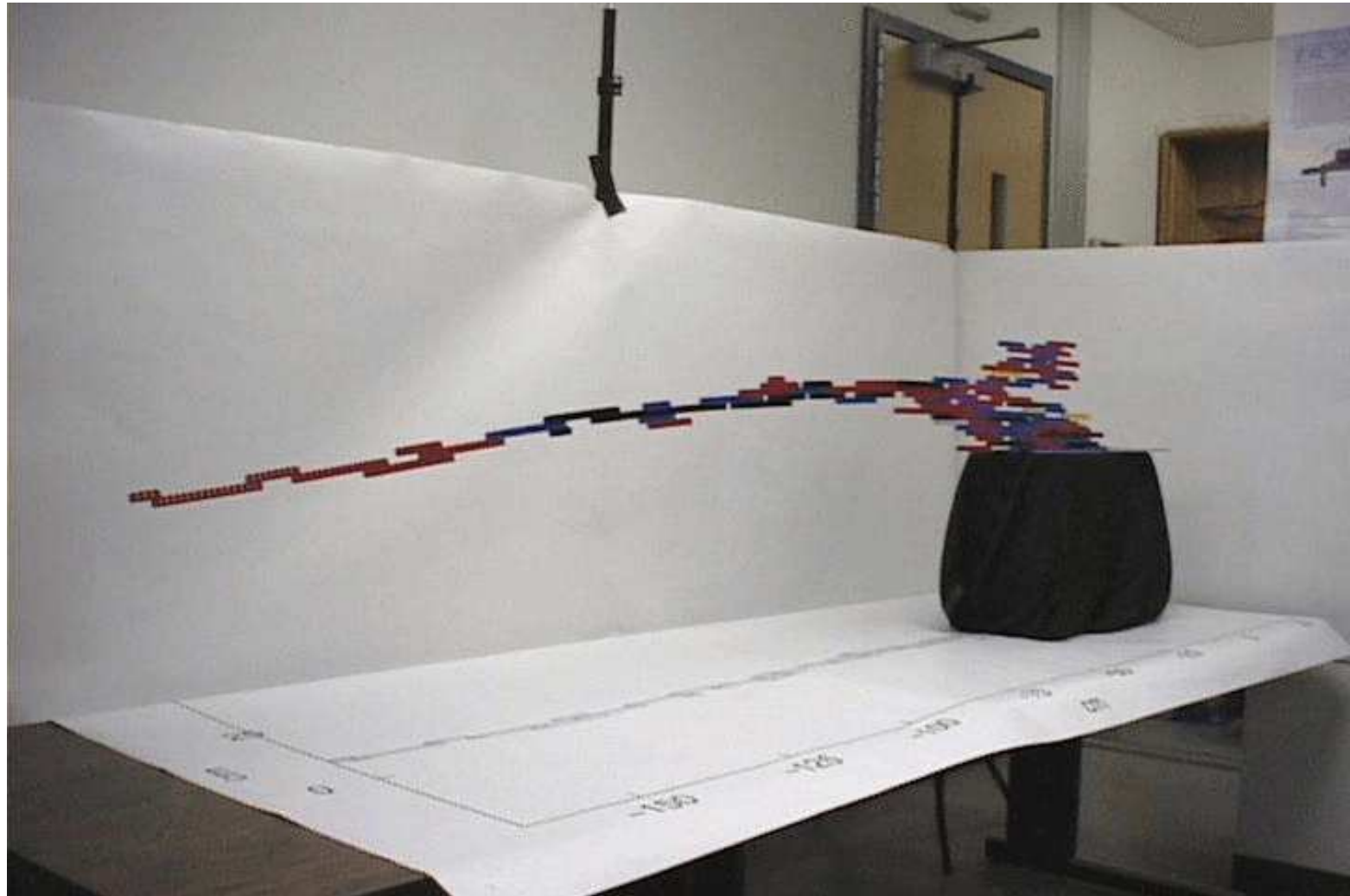
A bit of history

- Fogel, Owens and Walsh. Evolutionary programming: candidate solutions encoded as finite-state machines (simple computer programs).
- More recently: Koza. Genetic programming: solutions as tree-structured Lisp expressions.

GAs have ended up as the most popular technique but they do not exhaust the possibilities of the general idea of evolutionary computation.

An application

- Pablo Funes's evolved Lego structures



An application

- Fitness function of this cantilever is a simple distance measure.
- Evolved in simulation and then demonstrated in the real world.
- www.demo.cs.brandeis.edu/pr/projects.html#buildable

Other applications

- Gas pipeline layouts
- Aircraft design
- Scheduling
- Recurrent neural networks

Next time: Evol. Design II

- Search spaces, fitness landscapes, and other metaphors...
- What are the preconditions for a GA to work well?
- Evolutionary robotics: using GAs to design robot brains and bodies

References

- Bullock, S. (1999). Are artificial mutation biases unnatural?. In Floreano, D., Nicoud, J.-D., & Mondada, F. (Eds.), *Advances in Artificial Life: Fifth European Conference on Artificial Life (ECAL'99)*, Vol. 1674 of *Lecture Notes in Artificial Intelligence*, pp. 64–73. Springer, Berlin.
- Darwin, C. (1859). *The Origin of Species by Means of Natural Selection*. John Murray, London.
- Floreano, D., & Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In Cliff, D., Husbands, P., Meyer, J.-A., & Wilson, S. W. (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pp. 421–430. MIT Press / Bradford Books, Cambridge, MA.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
- Zaera, N., Cliff, D., & Bruten, J. (1996). (Not) evolving collective behaviors in synthetic fish. In Maes, P., Matarić, M., Meyer, J.-A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 635–644. MIT Press / Bradford Books, Cambridge, MA.