Evolution of Complexity in Real-World Domains

A Dissertation

Presented to

The Faculty of the Graduate School of Arts and Sciences

Brandeis University

Department of Computer Science

Jordan B. Pollack, Advisor

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

by

Pablo Funes

May, 2001

This dissertation, directed and approved by Pablo Funes's committee, has been accepted and approved by the Graduate Faculty of Brandeis University in partial fulfillment of the requirements for the degree of:

DOCTOR OF PHILOSOPHY

Dean of Arts and Sciences

Dissertation Committee:

Jordan B. Pollack, Dept. of Computer Science, Chair.

Martin Cohn, Dept. of Computer Science

Timothy J. Hickey, Dept. of Computer Science

Dario Floreano, ISR, École Polytechnique Fédérale de Lausanne

in memoriam

Everé Santiago Funes (1913-2000)

Acknowledgments

Elizabeth Sklar collaborated on the work on coevolving behavior with live creatures (chapter 3).

Hugues Juillé collaborated with the Tron GP architecture (section 3.3.3) and the novelty engine (section 3.3.7).

Louis Lapat collaborated on EvoCAD (section 2.9).

Thanks to Jordan Pollack for the continuing support and for being there when it really matters.

Thanks to Betsy Sklar, my true American friend. And to Richard Watson for the love and the focus on the real science. Also to all the people who contributed in one way or another, in no particular order: José Castaño, Adriana Villella, Edwin De Jong, Barry Werger, Ofer Melnik, Isabel Ennes, Sevan Ficici, Myrna Fox, Miguel Schneider, Maja Mataric, Martin Cohn, Aroldo Kaplan, Otilia Vainstok.

And mainly to my family and friends, among them: María Argüello, Santiago Funes, Soledad Funes, Carmen Argüello, María Josefa González, Faustino Jorge, Martín Levenson, Inés Armendariz, Enrique Pujals, Carlos Brody, Ernesto Dal Bo, Martín Galli, Marcelo Oglietti.

ABSTRACT

Evolution of Complexity in Real-World Domains

A dissertation presented to the Faculty of
the Graduate School of Arts and Sciences of
Brandeis University, Waltham, Massachusetts

by Pablo Funes

Artificial Life research brings together methods from Artificial Intelligence (AI), philosophy and biology, studying the problem of evolution of complexity from what we might call a constructive point of view, trying to replicate adaptive phenomena using computers and robots.

Here we wish to shed new light on the issue by showing how computer-simulated evolutionary learning methods are capable of discovering complex emergent properties in complex domains. Our stance is that in AI the most interesting results come from the interaction between learning algorithms and real domains, leading to discovery of emergent properties, rather than from the algorithms themselves.

The theory of natural selection postulates that generate-test-regenerate dynamics, exemplified by life on earth, when coupled with the kinds of environments found in the natural world, have lead to the appearance of complex forms. But artificial evolution methods, based on this hypothesis, have only begun to be put in contact with real-world environments.

In the present thesis we explore two aspects of real-world environments as they interact with an evolutionary algorithm. In our first experimental domain (chapter 2) we show how structures can be evolved under gravitational and geometrical constraints, employing simulated physics. Structures evolve that exploit features of the interaction between brick-based structures and the physics of gravitational forces.

In a second experimental domain (chapter 3) we study how a virtual world gives rise to co-adaptation between human and agent species. In this case we look at the competitive interaction between two adaptive species. The purely reactive nature of artificial agents in this domain implies that the high level features observed cannot be explicit in the genotype but rather, they emerge from the interaction between genetic information and a changing domain.

Emergent properties, not obvious from the lower level description, amount to what we humans call complexity, but the idea stands on concepts which resist formalization — such as difficulty or complicatedness. We show how simulated evolution, exploring reality, finds features of this kind which are preserved by selection, leading to complex forms and behaviors. But it does so without creating new levels of abstraction — thus the question of evolution of modularity remains open.

Contents

Chapter 1

Introduction

1.1 Artificial Life and Evolution of Complexity

1.1.1 How does complex organization arise?

The present state of the universe lies somewhere between two extremes of uniformity: perfect order, or zero entropy, which might have happened at the beginning of time (*big bang*), and total disorder, infinite entropy, which could be its final destiny (*heat death*). Somehow from these homogeneous extremes, diversity arises in the form of galaxies, planets, heavy atoms, life.

Analogous scenarios of emergence of organization exist within the scope of different sciences (*e.g.* formation of ecological niches, of human language, of macromolecules, of the genetic code, etc.). The study of *evolution of complexity* bears on these disciplines and aims at understanding the general features of such phenomena of "complexification": what is complexity, and what kinds of processes lead to it? [64, 77].

Biology in particular strives to explain the evolution of complex life forms starting from a random "primitive soup". Darwin's theory of natural selection is the fundamental theory that explains the changing characteristics of life in our planet, but contemporary Darwinism is far from complete, having only begun to address questions such as specialization, diversification and complexification.

Some theories of natural evolution argue that complexity is nothing but statistical error [57], whereas others propose that increasing complexity is a necessary consequence of evolution [63, 122]. The point of view of *universal Darwinism* [29] is that the same characteristics of life on earth that make it susceptible to evolutionary change by natural selection, can also be found on other systems, which themselves undergo evolutionary change. Plotkin [106] proposes the *g-t-r heuristics*[1] as the fundamental characteristic of evolutionary process. Three *phases* are involved:

1. ***g*** — Generation of variants

[1] An extension of the "generate-and-test" concept [102], with the additional token of iterated generation (*regeneration*) based on the previous ones.

2. ***t*** — Test and selection

3. ***r*** — Regeneration of variants, based on the previous ones

Some examples of systems subject to this kind of g-t-r dynamics are: life on earth, the mammal immune system (random mutation and selection of antigens), brain development (selection of neurons and synapses) and human language (selection of words and grammars) [28, 106].

1.1.2 Measuring Complexity

One of the problems with studying the mechanisms and history of complex systems is the lack of a working definition of *complexity*. We have intuitive notions that often lead to contradictions. There have been numerous attempts to define the complexity of a given system or phenomenon, usually by means of a *complexity measure* — a numerical scale to compare the complexity of different problems, but all of them fall short of expectations.

The notion of Algorithmic Information Content (AIC) is a keystone in the problem. The AIC or *Kolmogorov complexity* of a binary string is defined as the length of the shortest program for a Universal Turing Machine (UTM) whose output is the given string [19, 79, 127].

Intuitively, the simplest strings can be generated with a few instructions, e.g. "a string of 100 zeros"; whereas the highly complex ones require a program slightly longer than the string itself, e.g. "the string 0010111100101110000010100001000111100110". However, the minimal program depends on the encoding or "programming language" chosen; the difference between two different encodings being bound by a constant. Moreover, AIC is uncomputable. Shannon's entropy [121] is a closely related measure (it is an upper bound to AIC [80, 136]).

Further research on the matter of complexity measures stems from the notion that the most difficult, the most interesting systems are not necessarily those most complex according to algorithmic complexity and related measures. Just as there is no organization in a universe with infinite entropy, there is little to be understood, or compressed, on maximally complex strings in the Kolmogorov sense. The quest for mathematical definitions of complexity whose maximums lie somewhere between zero and maximal AIC [10, 53, 58, 82] has yet to produce satisfactory results. Bruce Edmonds' recent PhD thesis on the measurement of complexity [33] concludes that none of the measures that have been proposed so far manages to capture the problem, but points out several important elements:

1. Complexity depends on the observer.

   The complexity of natural phenomena *per se* can not be defined in a useful manner, because natural phenomena have infinite detail. Thus one cannot define the absolute or inherent complexity of "earth" for example. Only when observations are made, as produced by an acquisition model, is when the question of complexity becomes relevant: after the observer's model is incorporated.

2. "Emergent" levels of complexity

   Often the interactions at a lower level of organization (e.g. subatomic particles) result in higher levels with aggregate rules of their own (*e.g.* formation of molecules). A defining characteristic of complexity is a hierarchy of description levels, where the characteristics of a superior level *emerge* from those below it. The condition of emergence is relative to the observer; emergent properties are those that come from unexpected, aggregate interactions between components of the system.

   A mathematical system is a good example. The set of axioms determines the whole system, but demonstrable statements receive different names like "lemma", "property", "corollary" or "theorem" depending on their relative role within the corpus. "Theorem" is reserved for those that are difficult to proof and constitute foundations for new branches of the theory — they are "emergent" properties.

   A theorem simplifies a group of phenomena and creates a higher lever language. This type of re-definition of languages is typical of the way we do science. As Toulmin puts it, "The heart of all major discoveries in the physical sciences is the discovery of novel methods of representation, and so of fresh techniques by which inferences can be drawn" [135, p. 34].

3. Modularization with Interdependencies

   Complex systems are partially decomposable, their modules dependent on each other. In this sense, Edmonds concludes that among the most satisfactory measures of complexity is the *cyclomatic number* [33, p. 107] [129], which is the number of independent closed loops on a minimal graph.
   The cyclomatic number measures the complexity of an expression, represented as a tree. Expressions with either all identical nodes or with all different nodes are the extremes in an "entropy" scale, for they are either trivial or impossible to compress. The more complex ones in the cyclomatic sense are those whose branches are different, yet some subtrees are reused across branches. Such a graph can be reduced (fig. 1.1) so that reused subexpressions appear only once. Doing so reveals a network of entangled cross-references. The count of loops in the reduced graph is the cyclomatic number of the expression.

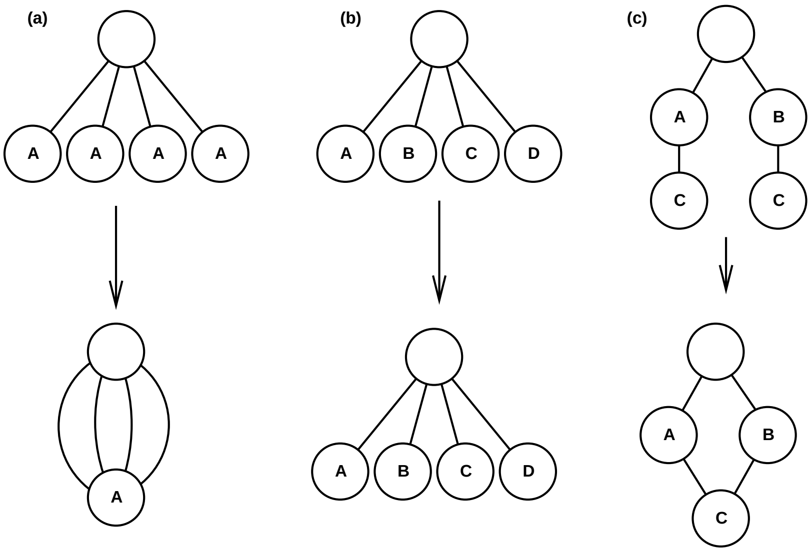


Figure 1.1: Cyclomatic complexity of expressions: (a) and (b) have cyclomatic number zero (no irreducible loops), although (a) is completely reducible and (b) completely irreducible. (c) has cyclomatic number one, because of the reuse of node C.

1.1.3 Artificial Life

Artificial Life (ALife) is a growing field that brings together research from several areas. Its subject is defined loosely as the study of "life as it could be" [87] as opposed to "life as it is" — which is the the subject of biology.

Work in ALife includes robots that emulate animal behaviors, agents that survive in virtual worlds, artificial evolution, reinforcement learning, autonomous robotics and so on. There is a continuing debate on this field regarding what the definition, methods and goals of Artificial Life are.

We propose that one of the fundamental goals of ALife research is to be a **constructive** approach to the problem of emergence of complexity. Not satisfied with a global description which describes the process through abstract elements, ALife should consider the question settled only when those elements have been formalized up to the point where they can be laid down in the form of a computer program and shown to work by running it.

Whereas evolutionary biology looks at the fossil record and tries to describe the evolution of life as a result of the g-t-r dynamics, Artificial Life research should aim at writing g-t-r programs that show how in fact artificial agents increase in complexity through the process, thus proving that natural complexity can be generated by this formal process.

1.1.4 Our Stance

The goal of this thesis is to show how the dynamics of computer-simulated evolution can lead to the emergence of complex properties, when combined with a suitable environment. We propose that one way to do this is to put evolutionary algorithms in contact with the real world, precisely because it was in this context that natural evolution led to the sophisticated entities conforming the biosphere.

Even though the characteristics that define “suitable environment” for evolution are unknown, we should be able to verify the theoretical predictions of the evolutionary hypothesis by placing artificial agents in the same kinds of contexts that produce complex natural agents.

The difficulty of measuring complexity makes it hard to study an evolutionary system acting on a purely symbolic domain, such as the *Tierra* experiments [112, 113]. Evolving real-world agents instead makes it easier to recognize solutions to difficult problems which are familiar to us, and at the same time creates an applied discipline, dealing with real problems.

We are deliberately staying away from a discussion about the different flavors of evolutionary algorithms (Genetic Algorithms, Genetic Programming, Multi-objective optimization and so on): all of them capture the fundamental ideas of the g-t-r model. Our aim is to reproduce the dynamics of natural evolution of complexity by *situating* artificial evolution within complex, reality-based domains. We are driven by the intuition that the most interesting results in our field have come not from great sophistication in the algorithm, but rather from the dynamics between g-t-r and interesting environments. Exciting results using coevolution [65, 109, 123, 131] for example, suggest that the landscape created by another adaptive unit is richer than a fixed fitness function.

Previous work in Artificial Life has already shown promising examples of evolving in real worlds. Here we implement two new approaches: the first one is evolving morphology under simulated physical laws, with simulated elements that are compliant to those found in reality, so as to make the results buildable. The second approach is to employ the concept of virtual reality to bring living animals into contact with simulated agents, in order to evolve situated agents whose domain, albeit simulated, contains natural life forms.

1.2 Interfacing an evolving agent with the Real World

Efforts to interface an adaptive agent with reality have created several subfields which study the problem from different perspectives.

1.2.1 Evolutionary Robotics

The field of Evolutionary Robotics (ER) starts with a fixed robot platform which has a computer brain connected to sensors and effectors. Different control programs or "brains" are tested by downloading them into the robot and evaluating its performance, either in the real robot or a simulated version [22, 37, 39, 67, 72].

Among the difficulties ER faces are,

- The robot is bounded by its physicality

  Evolution is limited by the pace and physicality of the robot. Making copies of a hardware robot is costly because they are not mass-produced. Also, the pace of time cannot be sped up.

- Design is costly

  The robot itself is designed by human engineers who engage in a costly process of designing, building, testing and repairing the robot. Commercial robots are available for research on a limited basis.

- Robotic Platform is fixed

  With a robot "body" whose morphology can not change, ER is limited to evolving control programs for the fixed platform. This represents a strong limitation, as we argue below, when compared to biological evolution where all behaviors are supported by morphology.

1.2.2 Reconfigurable Hardware

Reconfigurable hardware is a new field that evolves the hardware configurations of reconfigurable chips (FPGAs). The idea that an evolutionary algorithm can generate and test hundreds of hardware configurations very quickly is powerful and has produced exciting results [133, 134].

So far this type of work is limited to chips and thus can not be used to generate life-like creatures. The problems dealt with by evolved FPGA chips are electrical problems such as frequency filters, and occasionally brains to control robotic behaviors.

Interest is growing on the design of *self-reconfigurable robots* that can change morphology under program control [42, 76, 83, 139, 140]. This is a promising field that holds the exciting perspective of putting together features of both the reconfigurable hardware and evolutionary morphology fields.

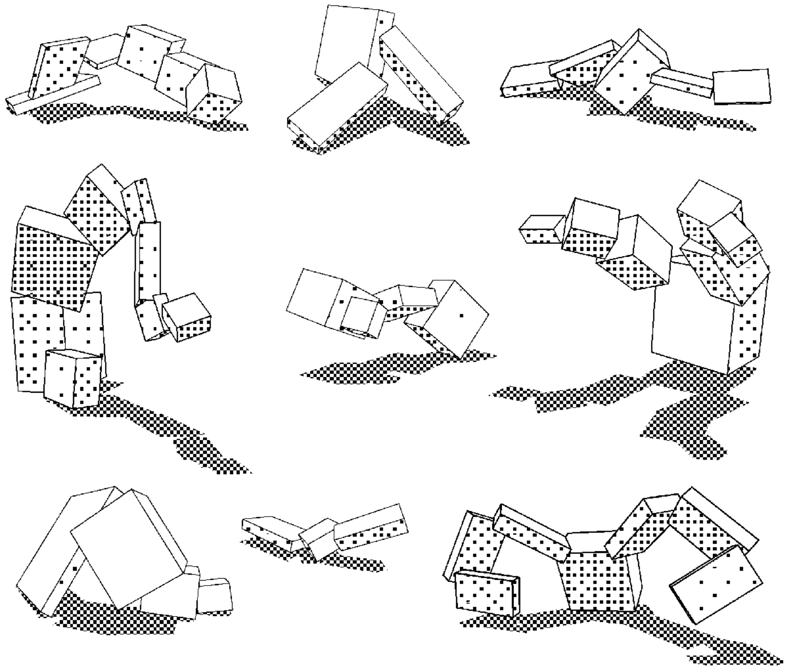


Figure 1.2: “Virtual Creatures” by K. Sims [123] have evolved morphology and behaviors. They behave according to some physical laws (inertia, action-reaction) but lack other reality constraints: blocks can overlap each other, movements are not generated by motors, etc.

1.2.3 Virtual Worlds

Virtual worlds are simulated environments with internal rules inspired at least in part by real physics laws. This type of environment has been fruitful for ALife research, for it allows quick implementation of reality-inspired behavior that can be visualized as computer animations [81, 101, 114, 123, 124].

The surreal beauty of some artificial agents in virtual worlds has had a profound impact in the field, most famously Karl Sims' *virtual creatures*, made of rectangular prisms, evolved life-like behaviors and motion under a careful physical simulation (fig. 1.2).

1.2.4 Simulated Reality

Simulating reality puts together Evolutionary Robotics and Virtual Worlds: at the same time one is dealing with the full complexity of physical reality, while not bounded by the laws of conservation of matter or the fixed pace of time. However, writing a full-fledged simulator is an impossibility, for reality has endless detail.

A heated discussion separates pro- and anti- simulation ALife research; the detractors of simulations [98, 133] argue that reality simply cannot be imitated, and that virtual agents adapted to simulated reality will evolve slower than real-time, fail to be transferable, or both. Advocates of simulation claim that by simulating only some aspects of reality one can evolve and transfer [71, 72].

1.3 Summary: Contributions of this Thesis

Here we investigate the *reality effect,* of which previous works in the field (Sims' virtual creatures, Thompson's FPGA's) are examples: evolution interacting with reality discovers emergent properties of its domain, builds complexity and creates original solutions, resembling what happens in natural evolution.

We investigate two complementary scenarios (a simulation that brings a computer brain out into the real world, and a video game which brings a multitude of natural brains into a virtual world) with two experimental environments:

1. Evolution of structures made of toy bricks (chapter 2). These are the main points discussed:

    - Evolutionary morphology is a promising new domain for ALife.
    - Adaptive designs can be evolved that are buildable and behave as predicted.
    - Principles of architectural and natural design such as cantilevering, counterbalancing, branching and symmetry are (re)discovered by evolution.
    - Recombination between partial solutions and change of use (*exaptation)* are mechanisms that create novelty and lead to the emergence of hierarchical levels of organization.
    - Originality results from an artificial design method that is not based upon predefined rules for task decomposition.
    - The potential for expanding human expertise is shown with an application — EvoCAD, a system where human and computer have active, creative roles.

2. Evolution of artificial players for a video-game (chapter 3). Main issues are:

    - Evolution against live humans can be done with a hybrid evolutionary scheme that combines agent-agent games with human-agent games.
    - The Internet has the potential of creating niches for mixed agent/human interactions that host phenomena of mutual adaptation.
    - Basic as well as complex navigation behaviors are developed as survival strategies.

- Coevolving in the real world is stronger than coevolving in an agent-only domain, which in turn is stronger than evolving against a fixed training set.
- Statistical methods are employed in order to analyze the results.
- Agents adapted to complex environments can exhibit elaborate behaviors using a simple reactive architecture.
- Human learning arises and can be studied from the interactions with an adaptive agent.
- An evolving population acts as one emergent intelligence, in an automated version of a *mixture of experts* architecture.

We conclude with a discussion on AI and the role of discovery and of interaction between learning algorithms, people and physical reality in the light of these results (chapter 4).

Altogether, we are elaborating on a new perspective of Artificial Life, conceived as one of the pieces on the question of evolution of complexity. The evolutionary paradigm explains complexification up to a certain point at least, but also shows that we are still far from a complete understanding of this phenomenon.

Chapter 2

Evolution of Adaptive Morphology

2.1 Introduction and Related Work

This chapter describes our work in evolution of buildable structural designs. Designs evolve by means of interaction with reality, mediated by a simulation that knows about the properties of their modular components: commercial off-the-shelf building blocks.

This is the first example of reality-constrained *evolutionary morphology*: entities that evolve under space and gravitational constraints imposed by the physical world, but are free to organize in any way. We do not consider movement; our aim is to show how complete structural organization, a fundamental concept for ALife, may begin to be addressed.

The resulting artifacts, induced by various manually specified fitness functions, are built and shown to behave correctly in the real world. They are not based in building heuristics or rules such as a human would use. We show how these artifacts are founded upon emergent rules discovered and exploited by the dynamic interaction of recombination and selection under reality constraints.

Weak search methods such as simulated evolution have great potential for exploring spaces of designs and artifacts beyond the limits of what people usually do. We present a prototype CAD tool that incorporates evolution as a creative component that creates new designs in collaboration with a human user.

Parts of this research have been reported on the following publications: [45–49, 110, 111].

2.1.1 Adaptive Morphology

Morphology is a fundamental means of adaptation in life forms. Shape determines function and behavior, from the molecular level, where the shape of a protein determines its enzymatic activity, to the organism level, where plants and animals adapt to specific niches by morphological changes, up to the collective level where organisms modify the environment to adapt it to their own needs.

In order to evolve adaptive physical structures we imitate nature by introducing a genetic coding that allows for both local modifications, with global effects (i.e. enlarging one component has only a local effect but may also result in shifting an entire subpart of the structure) and recombination, which spreads useful subparts of different sizes.

Even though the plasticity of life forms is far superior to a limited computer model such as ours, we are still able to see how the dynamic "evolutionary game" among a genetically-regulated family of organisms whose fitness comes from interaction with a complex environment, results in evolution of complexity and diversity leading to higher levels of organization.

2.1.2 ALife and Morphology

A deep chasm separates Artificial Life work that uses robotics models [22, 36, 37, 95] from the one in virtual worlds [81, 101, 114, 123, 124]. Robots today lack plasticity in their design; they need to be built by hand, molded using expensive methods. The number of generations and the number of configurations tested is several orders of magnitude smaller than those that can be reached with simulated environments. Evolutionary Robotics does not address morphology, although the idea was around from the beginning [23]. Experiments generally focus on evolving behaviors within a fixed morphology — a robotic "platform". Occasionally we see shape variables, but limited to a few parameters, such as wheel diameter or sensor orientation [26, 88, 96].

Evolution in Virtual Worlds on the other hand, is often morphological [81, 123]. Virtual worlds are constrained neither by the fixed "speed" of real time, nor by physical laws such as conservation of mass. The drawback is in the level of detail and the complexity of reality: simulating *everything* would require an infinite amount of computation.

The Evolution of Buildable Structures project aims at bridging the reality gap between virtual worlds and robotics by evolving agents in simulation under adequate constraints, and then transferring the results, constructing the "reality equivalent" of the agent.

Lego[1] bricks are popular construction blocks, commonly used for educational, recreation and research purposes. We chose these commercially available bricks because they have proven to be adequate for so many uses, suggesting that they have an appropriate combination of size, tightness, resistance, modularity and price. These characteristics led us to expect to be able to evolve interesting, complex structures that can be built, used and recycled.

[1]Lego is a registered trademark of the Lego group.

2.1.3 *Nouvelle* AI

One important contribution of the *nouvelle AI* revolution in the eighties was to deconstruct the traditional notion of reasoning in isolation. Brooks [15, 16] fought against the division of cognition in layers (perception – recognition – planning – execution – actuation) and instead proposed the notions of reactivity and situatedness: the phenomenon we call intelligence stems from a tight coupling between sensing and actuation.

In the same spirit of doing without layered approaches, we reject the notion of parameterizing the functional parts of a robotic artifact. The different parts of a body — torso, extremities, head — are not interesting if we establish them manually. By giving the evolutionary code full access to the substrate, the search procedure does without conventional human biases, discovering its own ways to decompose the problem — which are not necessarily those that human engineers would come up with.

Human cognition, as pointed out by Harvey [62] lacks the ability to design complex systems as a whole. Instead, we usually proceed by complexity reduction (the "divide and conquer" method). This is why the classic AI work took the layered approach that Brooks rejected so strongly. Perhaps the greatest strength of ALife methods such as artificial evolution is their ability to develop the organization and subparts together as a whole.

2.1.4 Artificial Design

The science of design usually conceives of AI as a set of tools for structuring the process, or planning, or optimizing [13, 20, 105]. Rarely does the computer explore a space of designs, and in doing so, it is generally following a set of precise rules, so the machine is doing little else than repeating a series of mechanical steps, faster than a human could. Creativity is usually considered to lay outside the realm of what computers can do.

Evolutionary Design (ED), the creation of designs by computers using evolutionary methods [11] is a new research area with an enormous potential. Examples of ED work are evolution of abstract shapes [12] or optimization of one part or component [20, 120]. The present work is different, for we are proposing to let the evolutionary process take care of the entire design process by means of recombination of available components and interaction with a physics simulation.

Inasmuch as Artificial Intelligence is an elusive concept— it seems that every new challenge that computers solve, becomes non-intelligent by definition[2], so is "artificial creativity". We claim that the present work is in fact a form of artificial creativity, albeit restricted, whose designs are unexpected, surprising, amusing — and they work.

[2] AI is "the study of how to make computers do things which, at the moment, people do better" [115, p. 3]

2.2 Simulating Bricks Structures

This section discusses our approach to the simulation of structures made of weakly joined bricks.

2.2.1 Background

Two kinds of simulation, Finite Elements from engineering and Qualitative Physics from computer science, have inspired our simulator of Lego brick structures.

Finite Element Modeling (FEM) is a structural mechanics technique for discretizing an object in order to analyze its behavior in the presence of stresses and holds [141]. The principle is to construct a network or "mesh" to model the piece as a discrete network and have the nodes communicate with their neighbors in order to cancel out all forces.

Qualitative Physics (QP) is a subfield of AI which deals with mechanical and physical knowledge representation. It starts with a logical representation of a mechanism, such as a heat pump [41] or a string [52], and produces simulations, or envisionments, of the future behavior of the mechanism.

QP simulations have not been used for evolutionary design, but they express an idea of great potential for reality-grounded evolution: not all aspects of the world need to be simulated to their fullest detail. Sometimes one can create an approximate model using *ad hoc* qualitative rules instead of the more complex equations of Newtonian physics.

2.2.2 Lego Bricks

The resistance of the plastic material (ABS - acrylonitrile butadiene styrene) of Lego bricks far surpasses the force necessary to either join two of them together or break their unions. This makes it possible to conceive a model that ignores the resistance of the material and evaluates the stress forces over a group of bricks only at their union areas. If a Lego structure fails, it will generally do so at the joints, but the actual bricks will not be damaged.

This characteristic of bricks structures makes their discretization for modeling an obvious step. Instead of imposing an artificial mesh for simulation purposes only (as FEM does), these structures are already made of relatively large discrete units. A first simplification is thus to ignore the physical characteristics of the bricks and study only those of their unions.

Our second simplification is to ignore bending effects. In standard structural analysis, the effects of stress are observed as deformation of the original shape of a body. Here strain deformations are ignored altogether.

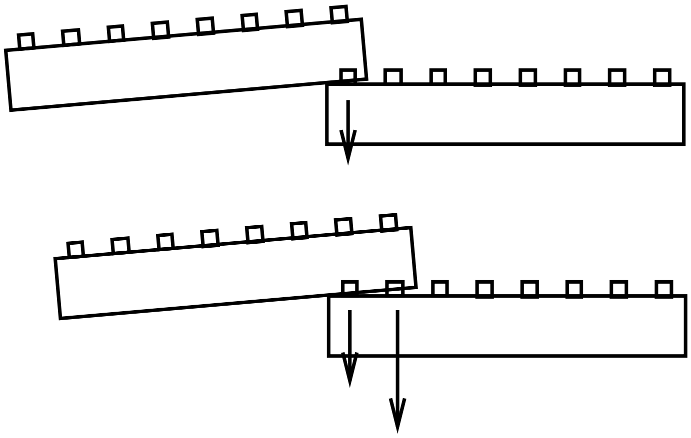


Figure 2.1: *Fulcrum effect*: a 2×1 union resists more than twice the load of a 1×1 because the second knob is farther away from the axis of rotation.

Joint size (ω) knobs	Approximate torque capacity (κ_ω) N-m$\times 10^{-3}$
1	12.7
2	61.5
3	109.8
4	192.7
5	345.0
6	424.0

Table 2.1: Estimated minimal torque capacities of the basic types of joints. Note: these values correct the ones on [48, table 1].

2.2.3 Joints in two dimensions

We began considering two-dimensional structures, assuming that all the bricks are of width 1, assembled in a plane. A fulcrum effect, which is the angular torque exerted over two joined bricks, constitutes the principal cause for the breakage of a stressed structure of Lego bricks. We designed our model around this idea, describing the system of static forces inside a complex structure of Lego bricks as a network of rotational joints located at each union between brick pairs and subject to loads (fig. 2.2).

Bricks joined by just one knob resist only a small amount of torque; bigger unions are stronger. The resistance of the joint depends on the number of knobs involved. We measured the minimum amount of stress that different linear (1×1, 2×1, 3×1, etc.) unions of brick pairs support (table 2.1).

From a structure formed by a combination of bricks, our model builds a network with joints of different capacities, according to the table. Each idealized joint is placed at the center of the area of contact between every pair of bricks. A margin of safety, set to 20% in our experiments, is discounted from the resistances of all joints in the structure, to ensure robustness in the model's predictions.

All forces acting in the structure have to be in equilibrium for it to be static. Each brick generates, by its weight, a gravitational force acting downwards. There may be other forces generated by external loads.

Each force has a site of application in one brick — each brick's weight is a force applied to itself; external forces also "enter" the structure through one brick — and has to be canceled by one or more reaction forces for that brick to be stable. Reaction forces can come from any of the joints that connect it to neighbor bricks. But the brick exerting a reaction force becomes unstable and has to be stabilized in turn by a reaction from a third brick. The load seems to "flow" from one brick to the other. Thus by the action-reaction principle, a load is propagated through the network until finally absorbed by a fixed body, the "ground".

The principle of propagation of forces described, combined with the limitations imposed to each individual joint, generates a set of equations (section 2.2.6). A solution means that there is a way to distribute all the forces along the structure. This is the principle of our simulator: *as long as there is a way to distribute the weights among the network of bricks such that no joint is stressed beyond its maximum capacity, the structure will not break.*

2.2.4 From 2- to 3-dimensional joints

In two dimensions, all brick unions can be described with one integer quantity — the number of knobs that join two bricks. Table 2.1 gives all the information needed to describe 2D brick joints. In the three dimensional case, brick unions are n-by-m rectangles. Two 2×4 bricks for example can be stuck together in 8 different types of joints: 1×1, 1×2, 1×3, 1×4, 2×1, 2×3, 2×4.

We know already, from the 2D case, how $n \times 1$ unions respond to forces acting along the x axis alone. A 2×1 union supports more than double the torque admitted by a 1×1, the reason being that the brick itself acts as a fulcrum (fig. 2.1). The distance from the border to the first knob is shorter than the distance to the second knob, resulting in a lower multiplication of the force for the second knob. This fulcrum effect does not happen when the force is orthogonal to the line of knobs. A 1×2 union can be considered as two 1×1 unions, or as one joint with double the strength of a 1×1 (fig. 2.3).

In other words, when torque is applied along a sequence of stuck knobs, the fulcrum effect will expand the resistance of the joint beyond linearity (as in table 2.1). But when

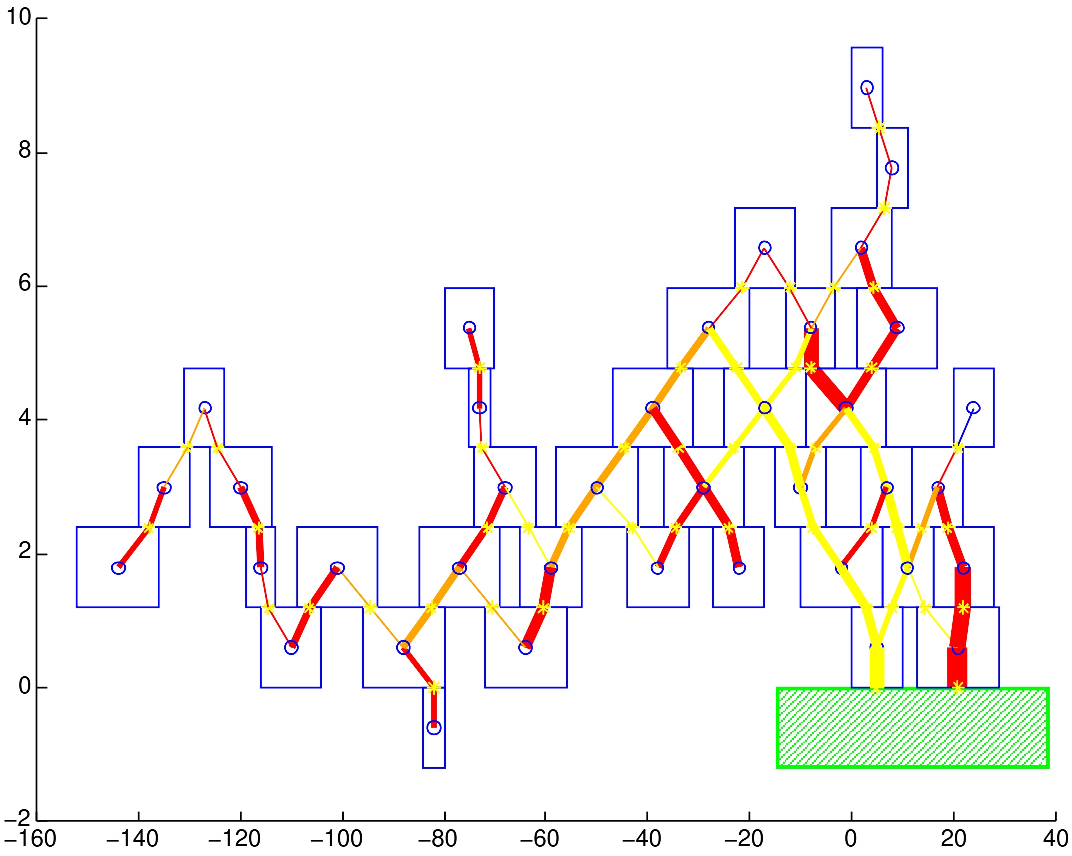


Figure 2.2: Model of a 2D Lego structure showing the brick outlines (rectangles), centers of mass (circles), joints (diagonal lines, with axis located at the star), and "ground" where the structure is attached (shaded area). The thickness of the joint's lines is proportional to the strength of the joint. A distribution of forces was calculated: highly stressed joints are shown in light color, whereas those more relaxed are darker. Note that the x and y axis are in different scales.

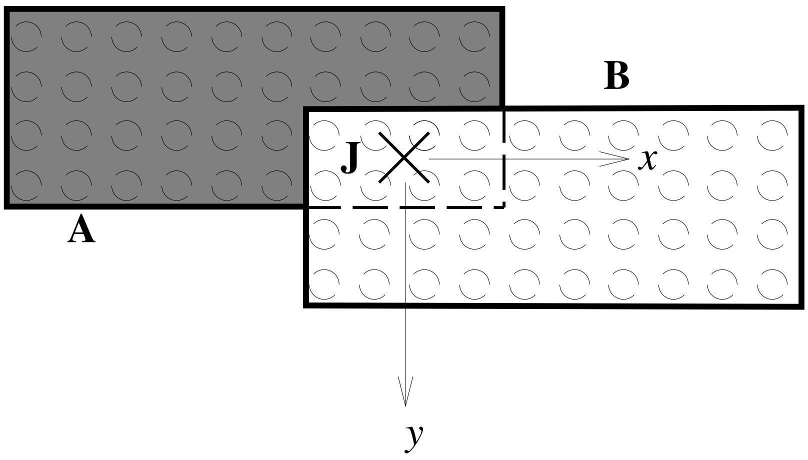


Figure 2.3: Two-dimensional brick joint. Bricks A and B overlap in a 4×2 joint J. Along x the joint is a double 4×1 joint. Along the y axis it is a quadruple 2×1-joint.

the torque arm is perpendicular instead, knob actions are independent and expansion is just linear.

We thus state the following *dimensional independence assumption*: Two bricks united by $n \times m$ overlapping knobs will form a joint with a capacity K_x along the x axis equal to m times the capacity of one n-joint and K_y along the y axis equal to n times the capacity of an m-joint.

To test the resistance of a composite joint to any spatial force f we separate it into its two components, f_x on the xz plane and f_y on the yz plane. These components induce two torques τ_x, τ_y. To break the joint either τ_x must be larger than K_x or τ_y larger than K_y.

If the dimensional independence hypothesis was not true, a force exerted along one axis could weaken or strengthen the resistance in the orthogonal dimension, but our measurements suggest that the presence of stress along one axis does not modify the resistance along the other. It is probably the case that the rectangular shape of the joint actually makes it stronger for diagonal forces, implying that dimensional independence is a conservative assumption. In any case, separating the components of the force has been a sufficient approximation for the scope of our experiments.

2.2.5 Networks of Torque Propagation

Our model for a 2D structure of bricks generates a network, called a Network of Torque Propagation (NTP) consisting of *nodes*, *joints* and *loads.*

- Each *node* represents a brick and its located at the brick's center of mass (circles in our figures).

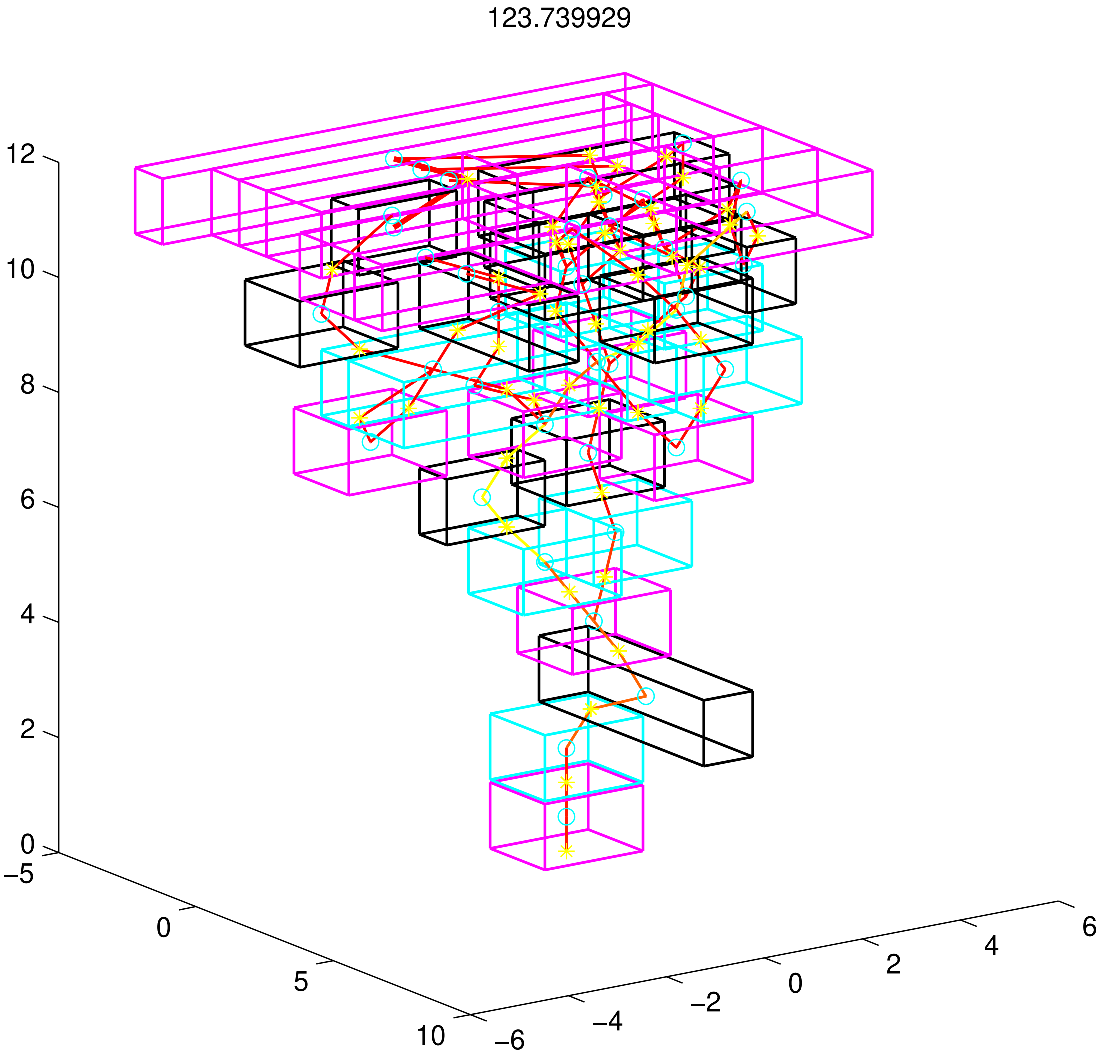


Figure 2.4: 3D Lego structure generated by our evolutionary process. The underlying physical model is shown.

- An additional node represents the ground.
- Each pair of locked bricks gives raise to a *joint*. The joint has an origin node, a destination node, an axis of rotation (located at the center of the area of contact between the bricks) and a maximum torque capacity (depending on the number of knobs involved). Joints are represented by lines in our figures, their axis of rotation by stars.
- *Loads* represent the forces acting on the network. Each has magnitude, direction, point of application, and entry node. For each brick, a force corresponding to its weight originates at the center of mass, is applied at the corresponding node, and points downwards. External forces may have any direction and their point of application is not necessarily the center of the brick.

Each force, either the weight of one of the bricks or an external load, has to be absorbed by the joints in the structure and transmitted to the ground. The magnitude of the torque exerted by each joint j must lie in the interval [$-K_j$, K_j], where K_j represents its maximum capacity as deduced from table 2.1.

By separating each 3D joint into two orthogonal and independent 2D joints, which receive the x and y components of each force, we can project an entire 3D network model of a brick structure into two orthogonal planes, xz and yz, generating two 2D NTP's that can be solved separately (figs. 2.4 and 2.5). Thus the problem of solving a 3D network is reduced to that of solving 2D networks.

2.2.6 NTP Equations

From our initial idea that forces propagate along a structure producing stresses in the form of torques, we have built an NTP, a network that has all the information needed to compute the possible paths along which the loads could "flow" and the torques they would generate along the way.

For each force F we consider the network of all the joints in the structure as a flow network that will transmit it to the ground. Each joint j can support a certain fraction α of such a force, given by the formula

$$\alpha_{j,F} = \max\left\{1, \left|\frac{K_j}{\delta(j,F)||F||}\right|\right\} \tag{2.1}$$

where K_j is the maximum capacity of the joint, $\delta(j,F)$ the distance between the line generated by the force vector and the joint, and $||F||$ the magnitude of the force. Thus if the torque generated is less than the joint maximum K, then $\alpha = 1$ (the joint fully supports F); otherwise α is K divided by the torque. The arm of the torque $\delta(j,F)$ can have a positive or negative sign depending on whether it acts clockwise or counterclockwise.

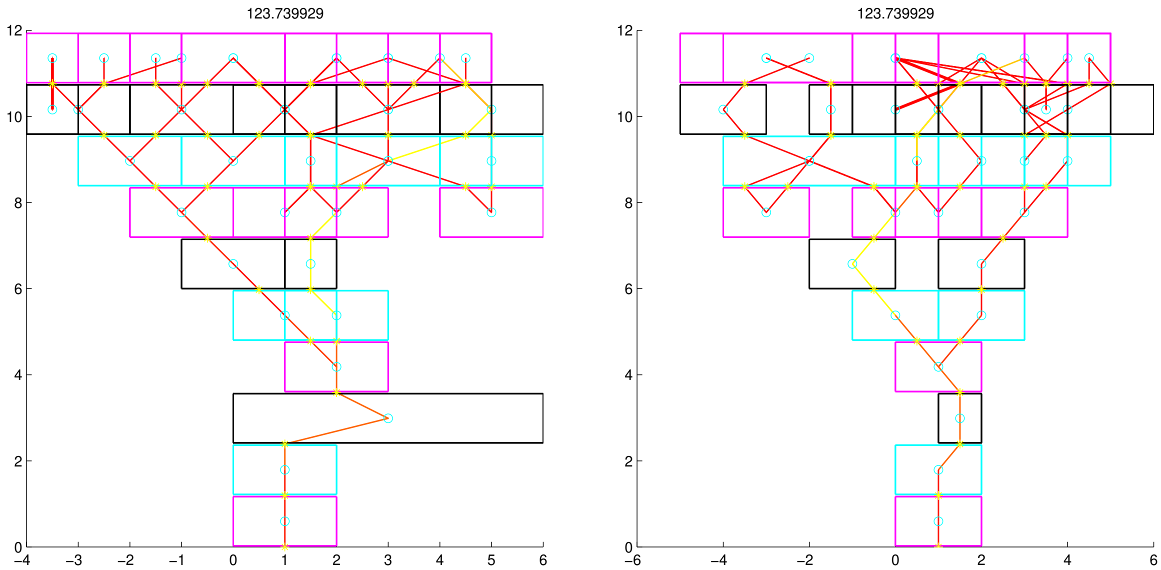


Figure 2.5: Projecting the 3D structure of fig. 2.4 to the *xz* and *yz* planes, two 2D networks are obtained that can be solved independently.

If one given force F is fixed and each joint on the graph is labeled with the corresponding $\alpha_{j,F}$ according to eq. 2.1, a network flow problem (NFP) [27] is obtained where the *source* is the node to which the force is applied and the *sink* is the ground. Each joint links two nodes in the network and has a capacity equal to $\alpha_{j,F}$. A net flow $|\phi_F| = 1$ represents a valid distribution of the force F throughout the structure: F can be supported by the structure if there is a solution to the NFP with a net flow of 1.

With more than one force, a solution for the entire network can be described as a set $\{\phi_F\}$ of flows, one for each force, all valued one. But as multiple forces acting on one joint are added, the capacity constraint needs to be enforced globally instead of locally, that is, the combined torques must be equal to or less than the capacity of the joint:

$$\left| \sum_F \phi_F(j)\delta(j,F)\,\|F\| \right| \leq K_j \tag{2.2}$$

This problem is not solvable by standard NFP algorithms, due to the multiplicity of the flow (one flow per force) and the magnification of magnitudes due to the torque arm δ (so the capacity of a joint is different for each load). Equation 2.2 is equivalent to a *multicommodity network flow problem* [2, ch. 17].

2.2.7 NTP Algorithms

Whereas the standard maximum network flow problem (single commodity) has well known polynomial-time solutions [27], multicommodity problems are much harder, and fall into the general category of linear programming. There is a fair amount of research on the multicommodity problem [3, 59, 68, 89] but the algorithms, based on Linear Programming, are exponential on the worst case.

Greedy Solver

Our initial approach for solving NTP problems was a greedy algorithm: Forces are analyzed one at a time. The push-relabel algorithm *PRF* by Cherkassky and Goldberg [21] is used to find a valid flow. Once a flow has been found it is fixed, and a remaining capacity for each joint (eq. 2.3) is computed that will produce a reduced network that must support the next force. A maximum flow is found for the second force with respect to the reduced network and so on for all forces.

$$K'_j = K_j - \phi_F(j)\delta(j,F)||F|| \tag{2.3}$$

This simple algorithm misses solutions, yet is quick, and thus we preferred it for time reasons to the more sophisticated solvers. With the greedy model, some solutions might be missed; but the ones found are good — so the structures evolve within the space of *provable* solutions, that is, those for which a greedy solution is found. This algorithm was particularly useful in the crane cases (sections 2.4, 2.6.3), where there is one special force, several orders of magnitude larger than the others. All experiments detailed here use this approach, except for the tree experiment (section 2.8) and EvoCAD (2.9), which employ the “embedded solver” explained below.

Multicommodity Solver

A second version of our Lego structure simulator incorporated a state-of-the-art multicommodity algorithm, by Castro and Nabona [18]. A special case of Linear Programming, these solvers have exponential order in the worst case, although with some luck they are faster on practical cases. We found this algorithm to be slower than the greedy version by a factor of 10 or more. The gain in accuracy did not compensate the loss in speed.

Embedded Solver

A third approach to the NTP problem was to incorporate the network flow into the representation of the structure. Thus structures and solutions evolve together: instead of using a network flow algorithm to find a flow, the flow is uniquely encoded in the genetic code of the structure, and is allowed to evolve along with it.

With a few modifications we extended the genotype to represent not only the position of the bricks, but also a unique flow for each force into a sink. With this, a structure can evolve along with the flows that represent a solution to the NTP problem.

As seen in the previous sections, a set $\{\phi_F\}$ of flows, one for each force, determines the total torque demanded from each joint in the structure (eq. 2.2). With the embedded solver, the evolutionary algorithm searches both the space of structure layouts and the space of flows at the same time. If the torques generated by the distribution of forces specified by the genotype exceed the joints' capacities, the structure is considered invalid.

Our representation for bricks structures (see section 2.3) is a tree graph whose nodes represent bricks. All descendants of a node are bricks which are physically in contact with the parent. In a structure there may be multiple paths from a brick to the ground, but genetically, there is a unique branch from each brick to the root. The root node is always a brick that rests on the ground, so all paths that follow the tree structure terminate on the ground. The following extensions to the genotype allowed us to evolve a structure along with the solution to the NTP problem:

1. Load flows only from descendant to ancestor

   Loads flow only down from descendants to parents. This defines the positive or negative sign of $\phi_F(j)$ for each joint and force. For the previous algorithms we had an undirected graph. Now the graph is strictly directed: for each brick pair a,b either joint $j(a,b)$ exists or $j(b,a)$, but not both.

2. Multiple trees rooted at grounds

   Instead of only one root, there can be multiple roots now situated at the grounds of the problem. Each load now has at least one possible path to flow to a sink, although it may or may not violate the joint's constraints.

3. "Adoptive" parents may also bear weight

   When two bricks happen to be physically linked, but neither of them is a descendant of the other, the first one[3] will become an "adoptive" parent, so the joint created flows from the lower-order brick to the higher-order.

4. Flow determined by joint size and weight vector.

   A weight parameter w_j was added to the representation of the joints. When a joint is created, w_j is initialized to 1, but then it may change by random mutation or by

[3]The tree is traversed in depth-first order. The descendants of a node are represented as a list, which determines the order of expansion, so there is a well-defined order in which bricks are laid down.

recombination. The flow $\phi_F(j)$ for each force and joint is determined by the joint size (number of knobs) and the flow weight, as follows:

Let x be a brick in the path of force F. The flow of F into x must equal its flow out of x, thus

$$F_x = \sum_a \phi_F(a,x) = \sum_b \phi_F(x,b) \tag{2.4}$$

The outgoing flow is uniquely determined by F_x and the proportion $\lambda(x,b)$ that goes to each parent b of x (either "adoptive" or "original").

For each brick b that is a parent of x, let $\omega(x,b)$ be the size (in knobs) of the joint $j(x,b)$ and $w(x,b)$ the encoded weight of the joint. Let $\Omega = \sum_{j(x,b)} \omega(x,b)$ and $W = \sum_{j(x,b)} w(x,b)$. For each joint now we define the proportion of total flow that follows each outgoing path as:

$$\lambda(x,b) = \frac{\omega(x,b)w(x,b)}{\Omega W} \tag{2.5}$$

which defines the behavior of all flows going through x:

$$\phi_F(x,b) = F_x \lambda(x,b) \tag{2.6}$$

With this configuration, the flow of a force through brick x is by default proportional to the size of the joint — stronger joints are asked to support proportionally more weight. But the genotype encodes weights $w(x,b)$ for each joint so the flow of the force can be redistributed.

5. Additional Mutations

   Two mutation operators were added to allow the structures to explore the space of possible flows:

   (a) Jump: A brick and its subtree of descendants is cut off from the original parent and becomes a descendant of one of its "adoptive" parents. This does not change the positions of any brick, but the directions of flow may change as bricks which were ancestors become descendants.

   (b) Redistribute Weight: A random joint's weight w_j is multiplied by a random number between zero and one resulting in a change of flow magnitudes.

This genotype extension was used for the tree experiments (section 2.8) and for EvoCAD (section 2.9). It does without any network problem solver and thus is much faster (by tenfold, approximately) at the cost of failing to approve many valid structures. In all, there was a speed benefit but *changes of function* were unlikely to happen (see section 2.6.2),

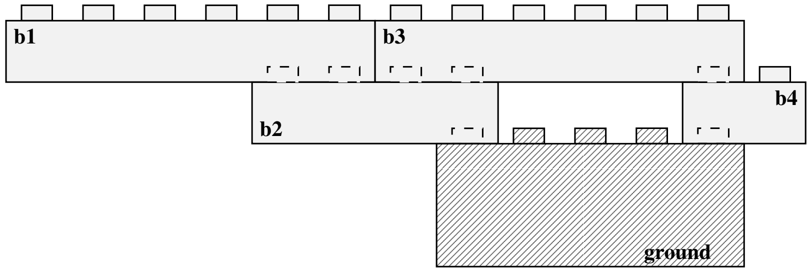


Figure 2.6: Sample structure with four bricks $b_1, \dots, b_4$ and a ground.

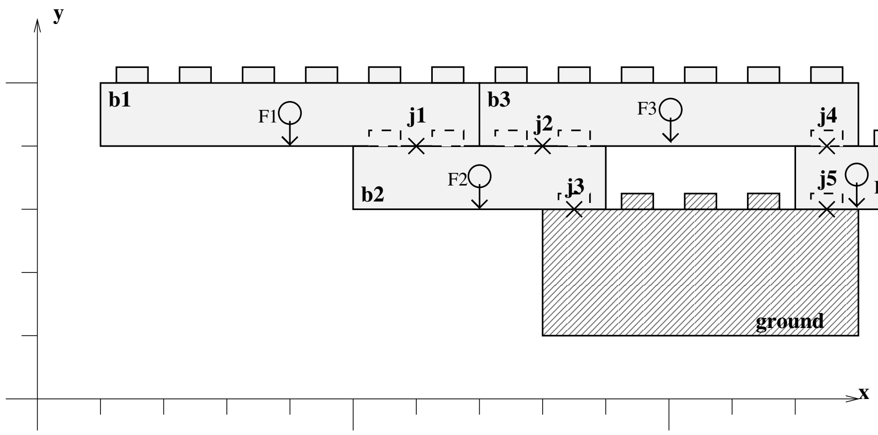


Figure 2.7: Loads and joints have been identified on the structure of fig. 2.6.

meaning that some of the richness of the dynamics between evolving agent and complex environment was lost when we embedded more of the environment inside the agent.

2.2.8 A Step-By-Step Example

In this section we build the NTP model for a sample brick structure in detail. We study a simple structure with four bricks and a ground (fig. 2.6).

In order to build the physical model, first we find the center of mass of all bricks (circles) and the center of the areas of contact between bricks (crosses), as shown on fig. 2.7. Each brick generates a force ($F_1, \dots, F_4$) and each area of contact, a joint ($j_1, \dots, j_5$). Adding an axis of reference, lists of loads (forces) and joints are generated (tables 2.2 and 2.3). For the sake of simplicity the x and y axis are in "Lego units": the width of a Lego unit is lw = 8 mm and the height, lh = 9.6 mm.

From the layout we generate a graph that represents the connectivity of the structure (fig. 2.8). Bricks and ground generate nodes on the graph and joints generate edges.

We consider initially what the situation is for the first load alone (F_1). This force is

n	position	direction	source	magnitude
F_1	(4,4.5)	(0,-1)	b_1	$6\beta G$
F_2	(7,3.5)	(0,-1)	b_2	$4\beta G$
F_3	(10,4.5)	(0,-1)	b_3	$4\beta G$
F_4	(13,3.5)	(0,-1)	b_4	$2\beta G$

Table 2.2: Loads obtained from fig. 2.7. β = weight of a Lego brick unit (0.4 g). G = gravitational constant.

n	nodes	position	knobs	max. torque K
j_1	(b_1, b_2)	(6,4)	2	κ_2
j_2	(b_2, b_3)	(8,4)	2	κ_2
j_3	(b_2, G)	(8,3)	1	κ_1
j_4	(b_3, b_4)	(12.5,4)	1	κ_1
j_5	(b_4, G)	(12.5,3)	1	κ_1

Table 2.3: Joints generated from fig. 2.7. The torque resistances κ_1, κ_2 are listed on table 2.1.

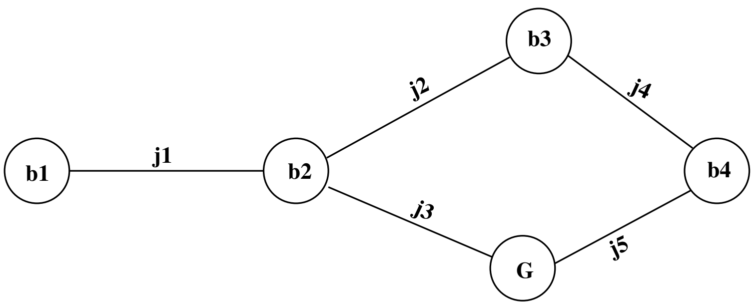


Figure 2.8: Graph generated by the structure of fig. 2.7.

Joint	Force	arm length (δ)	relative capacity (α)	sign
j_1	F_1	2 lw	$\frac{\kappa_2}{2\cdot 6\,\mathrm{lw}\beta\mathrm{G}}$	-1
j_2	F_1	4 lw	$\frac{\kappa_2}{4\cdot 6\,\mathrm{lw}\beta\mathrm{G}}$	-1
j_3	F_1	4.5 lw	$\frac{\kappa_1}{4.5\cdot 6\,\mathrm{lw}\beta\mathrm{G}}$	-1
j_4	F_1	8.5 lw	$\frac{\kappa_1}{8.5\cdot 6\,\mathrm{lw}\beta\mathrm{G}}$	-1
j_5	F_1	8.5 lw	$\frac{\kappa_1}{8.5\cdot 6\,\mathrm{lw}\beta\mathrm{G}}$	-1

Table 2.4: Capacities of the example network with respect to load F_1. Each joint can support a fraction of the load equal to the torque capacity of the joint divided by the torque exerted by that particular force at that joint, which in turn is the arm length multiplied by the magnitude of the force. lw = width of a Lego brick = 0.8 mm.

Joint	Force	Flow $\phi_{F_1}(j)$	torque (lwβG)	residual capacity(*)
j_1	F_1	1.0	$-1.0\cdot 2\cdot 6$	$[-33,57]$
j_2	F_1	0.3	$-0.3\cdot 4\cdot 6$	$[-37.8,52.2]$
j_3	F_1	0.7	$-0.7\cdot 4.5\cdot 6$	$[-1.1,38.9]$
j_4	F_1	0.3	$-0.3\cdot 8.5\cdot 6$	$[-4.7,35.3]$
j_5	F_1	0.3	$-0.3\cdot 8.5\cdot 6$	$[-4.7,35.3]$

Table 2.5: Greedy solver: Residual joint capacities for the sample structure, after force F_1 has been distributed according to fig. 2.9. (*) Assuming $\kappa_2 = 45$, $\kappa_1 = 20$.

originated by the mass of brick number one, and so it points downwards, its magnitude being equal to the weight of a Lego brick of width six (= 6 βG, where β = 0.4 g is the per-unit weight of our Lego bricks, and G the earth's gravitational constant). According to equation 2.1, the capacity of each joint with respect to this particular load is the magnitude of the load, multiplied by the torque's arm and divided by the capacity of the joint (table 2.4). The value of the sign is 1 if the rotation is clockwise and -1 if counterclockwise.

With the true values[4] for κ_1 and κ_2, the capacities of all joints in the example are far greater than the light forces generated by this small structure. To illustrate distribution of force we use fictitious values for the constants. Assuming $\kappa_1 = 20\,\mathrm{lw}\beta\mathrm{G}$ and $\kappa_2 = 45\,\mathrm{lw}\beta\mathrm{G}$, the capacities of joints $j_1,\ldots,j_5$ relative to load F_1 are respectively 1, 1, $\frac{20}{27}$, $\frac{20}{51}$ and $\frac{20}{51}$, leading to the network flow problem (and solution) on fig. 2.9. Each edge was labelled with the capacity and (parenthesized) a solution.

The solution to this flow problem could have been obtained by a maximum flow algorithm. A greedy solver would reduce now the network, computing a "remaining capacity" for each joint (table 2.5). The stress on joint j_2 for example, is equal to $-0.3\cdot 4\cdot 6\,\mathrm{lw}\beta\mathrm{G}$

[4]According to table 2.1, and assuming $\mathrm{G} = 9.8m/s^2$, the values of $\kappa_1,\ldots,\kappa_6$ are respectively: 405, 1960, 3500, 6144, 11000 and 13520 lwβG.

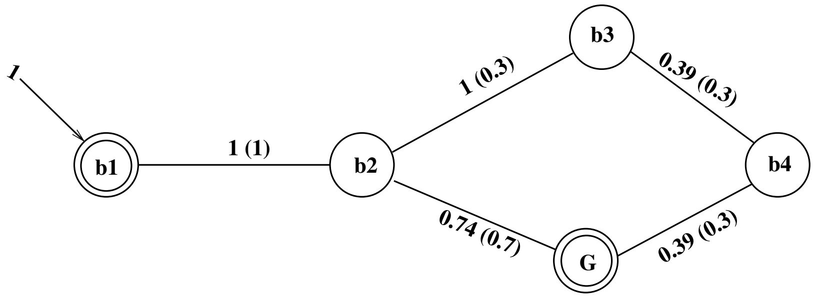


Figure 2.9: Network flow problem generated by the weight of brick b_1 on the sample structure of fig. 2.7, assuming $\kappa_1 = 20$ and $\kappa_2 = 45$ lwβG. The source is b_1 and the sink G. Each node is labelled with a capacity, and (in parenthesis) a valid flow is shown: $\phi_1(1,2) = 1$, $\phi_1(2,3) = 0.3$, $\phi_1(3,4) = 0.3$, $\phi_1(4,G) = 0.3$, $\phi_1(2,G) = 0.7$.

joint	force	arm length (lw)	magnitude (lwβG)	sign	capacity (w.r.t. F_2) (see table 2.9)
j_1	F_2	1	4	1	1
j_2	F_2	1	4	-1	1
j_3	F_2	1.5	6	-1	$\frac{1.1}{6}$
j_4	F_2	5.5	22	-1	$\frac{4.7}{22}$
j_5	F_2	5.5	22	-1	$\frac{4.7}{22}$

Table 2.6: Greedy solver: capacities of the joints in the sample structure, with respect to force F_2, after the loads resulting from F_1 have been subtracted.

(counterclockwise). If the initial capacity of j_2 was $[-\kappa_2, \kappa_2] = [-45, 45]$, the reduced capacity (according to eq. 2.3) would be $[-37.8, 52.2]$. So when a flow network for F_2 is generated, the reduced capacities of joints are used, incorporating the effects of the previous load. (table 2.6 and figure 2.10). In this example, there is no solution, so in fact the structure could not be proved stable.

For the multicommodity solver, all forces are considered simultaneously. The capacities of each joint become boundary conditions on a multicommodity network flow problem. For example, we can write down the equation for joint number two by generating a table of all forces and their relative weights for this particular joint (table 2.7).According to the table, and per equation 2.2, if $\phi_1, \ldots, \phi_4$ are the flow functions of forces $F_1, \ldots, F_4$, the boundary condition for joint two is:

$$|-24\,\phi_1(2,3) - 4\,\phi_2(2,3) + 12\,\phi_3(3,2) + 10\,\phi_4(3,2)| \leq \kappa_2 \tag{2.7}$$

a solution to this problem is a set of four flows $\phi_1, \ldots \phi_4$, each one transporting a magnitude of one from the origin of each force (F_i originates at b_i in our example) into the sink G,

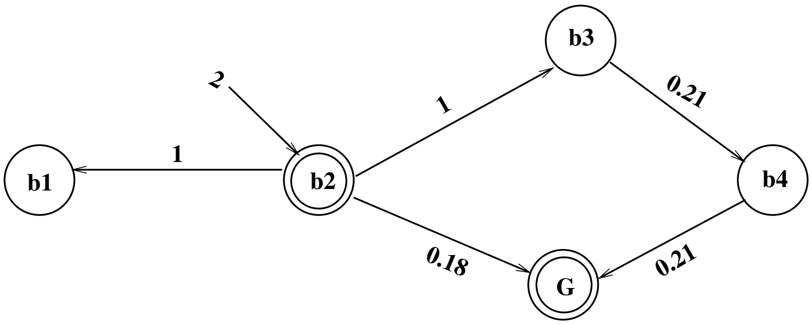


Figure 2.10: Greedy solver: NFP problem for the second force.

joint	force	arm length (lw)	magnitude (lwβG)	sign
j_2	F_1	4	$6 \cdot 4$	-1
j_2	F_2	1	$4 \cdot 1$	-1
j_2	F_3	2	$6 \cdot 2$	1
j_2	F_4	5	$2 \cdot 5$	1

Table 2.7: Relative weights of the forces from fig. 2.7 as they act on joint number two.

that also satisfies five boundary equations, analogous to eq. 2.7, one per joint. A multicommodity flow algorithm searches the space of all possible flows, using linear programming techniques, looking for such a solution.

Finally, using the embedded solver would mean that the genotype pre-specifies a unique direction of flow, and a weight for all joints, as in table 2.8 for example. Figure 2.11 shows the resulting DAG which determines all four flows (table 2.9) and thus the total torque for all the joints. Whereas in the greedy example we had the weight F_1 of brick b_1 flowing in part via $b_2 \to G$ (30%) and in part via $b_2 \to b_3 \to b_4 \to G$ (70%), in this embedded solver example, the only route allowed for the genotype is F_1 is via $b_2 \to G$. Again, with the

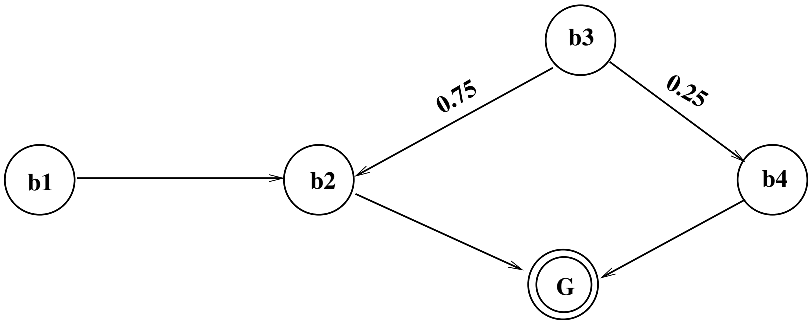


Figure 2.11: DAG determined by the genotype of a structure using the embedded solver approach.

joint	direction	weight (w)	knobs (ω)	ωw
j_1	$b_1 \to b_2$	1	2	2
j_2	$b_3 \to b_2$	1.5	2	3
j_3	$b_2 \to G$	0.75	1	0.75
j_4	$b_3 \to b_4$	1	1	1
j_5	$b_4 \to G$	2	1	2

Table 2.8: Embedded solver: the genotype specifies direction of flow and a random weight for each joint. Together with the number of knobs of in the joint, these specify the percentage of flow in each direction. In this example, brick b_3 has two outgoing joints, to bricks b_2 and b_4, with ωw values of 3 and 1, respectively. This means that 75% of any loads going through b_3 will pass on to b_2 and the remaining 25% will rest on b_4.

flow	value
ϕ_1	$b_1 \to b_2 \to G$
ϕ_2	$b_2 \to G$
ϕ_3	$\frac{3}{4}b_3 \to b_2 \to G + \frac{1}{4}b_3 \to b_4 \to G$
ϕ_4	$b_4 \to G$

Table 2.9: Flows generated by the embedded solution.

values for κ_1, κ_2 used in this example, the weight on joint j_3 is excessive so the structure is not stable.

2.3 Evolving Brick structures

Our representation to evolve Lego structures borrows the tree mutation and crossover operators from genetic programming (GP) [85]. We implemented tree representations of 2D and 3D Lego structures. Each node of the tree represents a brick and has a size parameter, indicating the size of the brick, and a list of descendants, which are new bricks physically attached to the parent. Each descendant node has positional parameters that specify the position of the new brick relative to the parent.

2.3.1 Coding for 2D and 3D structures

In the first, 2D version of this work [46], each brick node had a size type parameter (4, 6, 8, 10, 12 or 16, corresponding to the Lego bricks of size 1×4 through 1×16) and four potential descendants, each one representing a new brick linked at one of its four corners (lower left, lower right, upper right, upper left). Each non-nil descendant had a 'joint size' parameter indicating the number of overlapping knobs in the union.

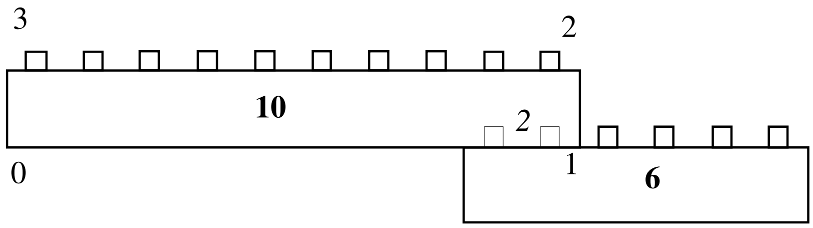


Figure 2.12: Example of 2D genetic encoding of bricks (eq. 2.8).

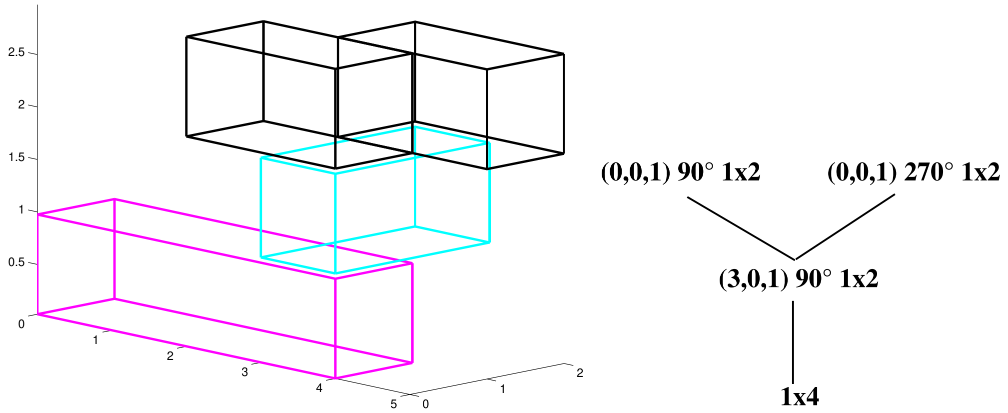


Figure 2.13: Model and tree representation for a few Lego bricks (eq. 2.9).

Fig. 2.12 represents a 10-brick with its 4 joint sites labeled 0, 1, 2, 3, that is linked to a 6-brick by two overlapping knobs. The corresponding tree could be written in Lisp-like notation as

$$(10\ \ \text{nil}\ \ (2\ \ (6\ \ \text{nil}\ \ \text{nil}\ \ \text{nil}))\ \ \text{nil}\ \ \text{nil}) \tag{2.8}$$

For 3D structures we added more size types to incorporate bricks other than $1 \times n$ (the table experiment in section 2.4.2 had sizes 1×2, 1×4, 1×6, 1×10, 1×12, 1×16, 2×2 and 2×4), and used a list of descendants, each one representing a new brick to be plugged into the parent. Each descendant brick has 3 parameters: The *(x, y, z)* coordinates of the new brick (relative to its parent, so for a descendant of an $n \times m$ brick, $0 \leq x < n$, $0 \leq y < m$ and $z \in$ {-1,1}); a rotation parameter that specifies the orientation of the descendant relative to the parent (0^o, 90^o, $180^o\, or\, 270^o$), and the size of the descendant. As an example, the structure in fig. 2.13 can be codified as

$$(1 \times 4\,(((3,0,1)\,90^o\,(1 \times 2\,(((0,0,1)\,90^o\,(1 \times 2\,\text{nil})(((1,0,1)\,270^o\,(1 \times 2\,\text{nil})))))))) \tag{2.9}$$

2.3.2 Mutation and Crossover

Mutation operates by either random modification of a brick's parameters (size, position, orientation) or addition of a random brick.

The crossover operator involves two parent trees out of which random subtrees are selected. As in GP, the offspring generated has the first subtree removed and replaced by the second.

After mutation or crossover operators are applied, a new, possibly invalid specification tree is obtained. The result is expanded one node at a time and overlapping is checked. Whenever an overlap is found the tree is pruned at that site. With this procedure, a maximum spatially valid subtree is built from a crossover or mutation. Branches that could not be expanded are discarded.

The following mutation of 2.9, for example, is illegal because two bricks would share the same physical space (z = -1 after the second brick means that the third one goes below it, but the first brick is already there).

$$(1\times 4\,(((3,0,1)\,90^o\,(1\times 2\,(((0,0,-1)\,90^o\,(1\times 2\,\mathrm{nil})(((1,0,1)\,270^o\,(1\times 2\,\mathrm{nil}))))))))) \tag{2.10}$$

The tree will be pruned then at the site, yielding just three bricks

$$(1\times 4\,(((3,0,1)\,90^o\,(1\times 2\,(((1,0,1)\,270^o\,(1\times 2\,\mathrm{nil})))))))) \tag{2.11}$$

Once a valid tree has been obtained, the simulator is called to test for stability. Fitness is evaluated and the new individual is added to the population.

2.3.3 Evolutionary Algorithm

We use a plain steady-state genetic algorithm, initialized with a population of one single brick. Through mutation and crossover, a population of 1000 individuals is generated and then evolved:

1. `While maximum fitness < Target fitness`
2. `Do`
   (a) `Randomly select mutation or crossover.`
   (b) `Select 1 (for mutation) or 2 (for crossover) random individual(s) with fitness proportional probability.`
   (c) `Apply mutation or crossover operator`

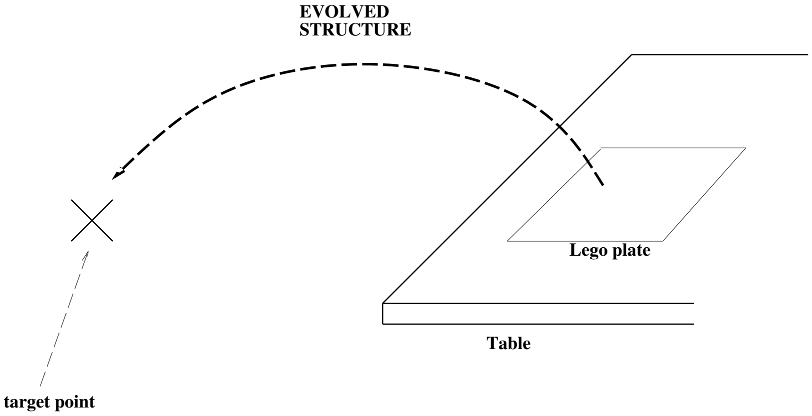


Figure 2.14: Lego bridge experimental setup: The structure starts on a Lego plate affixed to a table and has to reach a target point supporting its own weight.

(d) `Generate physical model and test for gravitational load`

(e) `If the new model will support its own weight`

i. `Then replace a random individual with it (chosen with inverse fitness proportional probability).`

2.4 Initial Experiments

2.4.1 Reaching a target point: Bridges and Scaffolds

In our first experiments we conceived a Lego plate affixed to a table (fig. 2.14) and evolved 2D structures to reach a target point, using as fitness function a normalized distance to the target point,

$$Nd(S,T) = 1 - \frac{d(S,T)}{d(0,T)} \tag{2.12}$$

(where S is the structure, T the target point and d the euclidean distance).

With a target point T located horizontally and away from the plate we generated a Lego bridge (figs. 2.2 and 2.15). Moving T to a remote position we obtained the "long bridge" (fig. 2.20), and putting T below we generated a descending structure, a "scaffold" (fig. 2.16).

External Loads: Horizontal crane arm

With a two-step fitness function that gives one point for reaching a target point as in eq. 2.12 and, if reached, additional points for supporting an external weight hanging from the

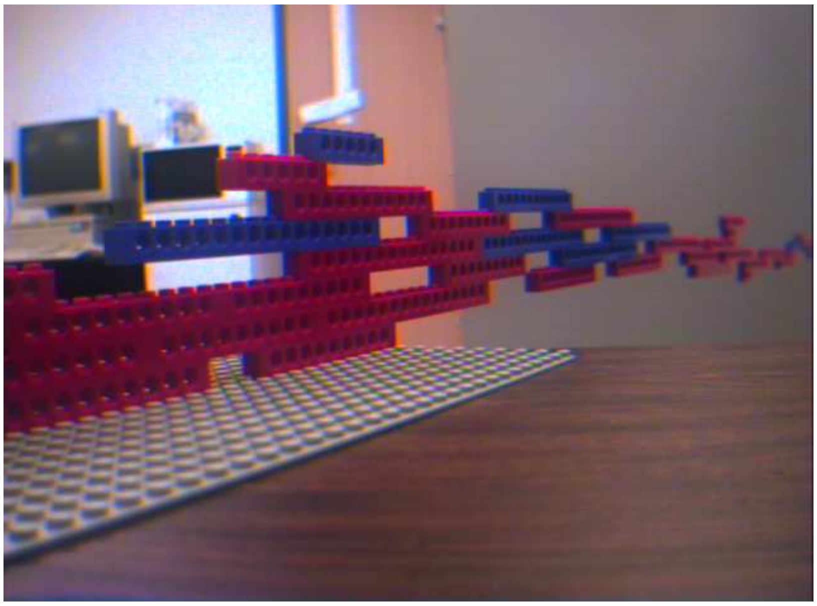


Figure 2.15: The Lego bridge defined by the scheme of fig. 2.2, built on our lab table.

last brick, we evolved a crane arm, fig. 2.17. The diagonal crane arm discussed in section 2.6.3 carries a fixed load as far as possible, rather than keeping it at a fixed place.

2.4.2 Evolving three-dimensional Lego structures: Table experiment

To evolve a Lego table we started with a fixed plate as in fig. 2.14, and wanted to obtain a table 10 bricks tall, with a support surface of 9×9 and capable of supporting a weight of 50 g located anywhere over this surface. There were four objectives to fulfill:

1. The height of the structure must be as required.
2. The surface most cover the target area.
3. The desired weight has to be supported all over the surface.
4. All other conditions met, a minimal number of bricks should be used.

To cover all the objectives we wrote a step fitness function giving between 1 and 2 points for the first objective partially fulfilled, between 2 and 3 for the first objective completed and partial satisfaction of the second, and so on. With this setup, the algorithm built upward first, then broadened to cover the surface, later secured that all points of the surface supported a load of 50g and finally tried to reduce the number of bricks to a minimum.

One of the solutions we obtained is shown in figs. 2.4 and 2.5; fig 2.18 is a photo of the finished table.

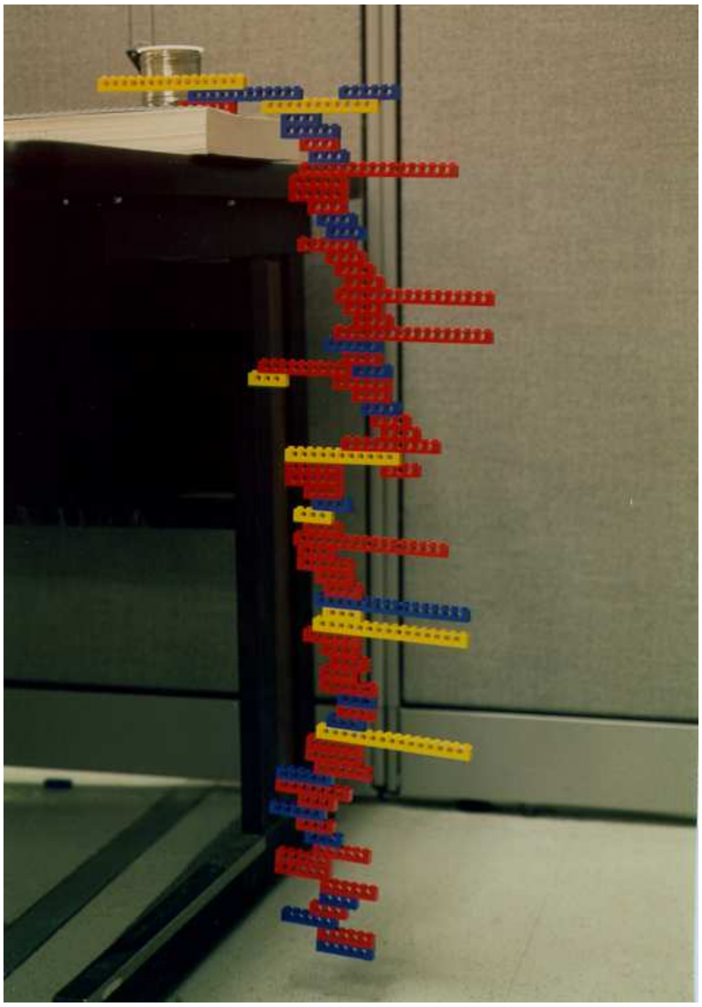


Figure 2.16: Scaffold.

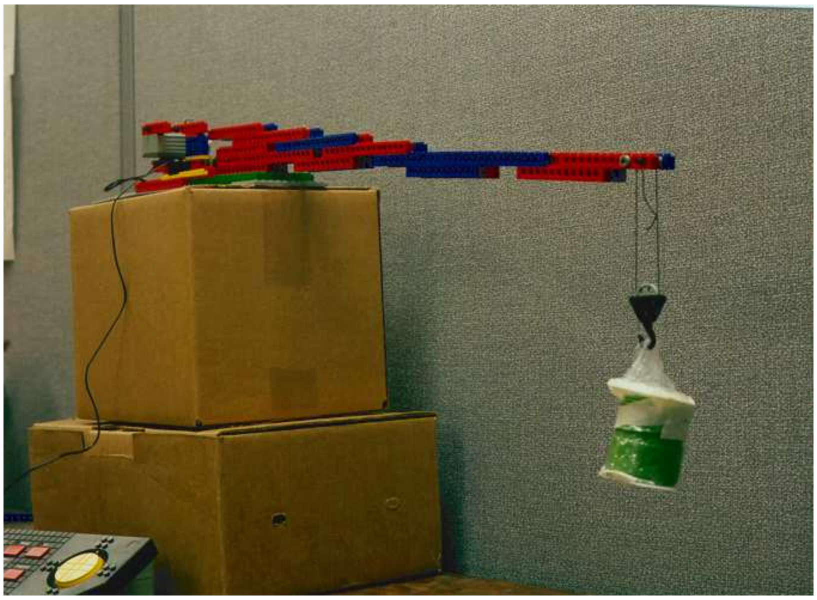


Figure 2.17: Crane with evolved horizontal crane arm.

Figure 2.18: Lego table as specified by the diagram of fig. 2.4, holding a 50g weight.

Problems defining the fitness function

A first attempt to evolve a table failed to satisfy objective 2 (covering the entire surface). One problem with our representation is that the distance between genotype and phenotype is big: mutations are likely to produce large changes on the structure, so detailed optimization is hard to obtain. Also, not having 1×1 bricks complicates matters (our set of Lego did not include those at the time). Finally, there was little selective pressure as fitness values between 1.9 and 2.0 where nearly identically selected. In the final run we expanded the raw fitness exponentially in order to add pressure (so for example the fitness value of 123.74 in fig. 2.4 corresponds to a raw fitness of $4.8 = \ln(123.74)$), but this did not solve the problem of full coverage. For the successful run pictured above, objective 2 was redefined as "covering at least 96% of the target area".

The use of stepped fitness functions is not ideal; Pareto front techniques [55, ch. 5] should improve performance in multiobjective problems such as this one.

2.5 Smooth Mutations and the Role of Recombination

A second version of the mutation operators introduced "smooth" mutations, which have better probabilities of producing small changes in the structure. The original mutation was simply "replace a randomly selected brick with a random single brick". The smooth mutations are of four kinds:

1. Brick Grow

   A randomly selected brick is enlarged, either to the left or right, to the nearest valid brick size.

2. Brick Shrink

   A randomly selected brick is shrunk, either from the left or the right, to the next smaller valid brick size.

3. Brick Shift/Brick Shift & Push

   A randomly selected brick shifts position, either to the left or right, by one knob. Any descendant bricks will either be "pushed" along in the same direction, or remain in their original positions.

4. Brick Add

   A brick is randomly selected. A random knob is chosen within that brick, above or below, and a new random brick is added at that point.

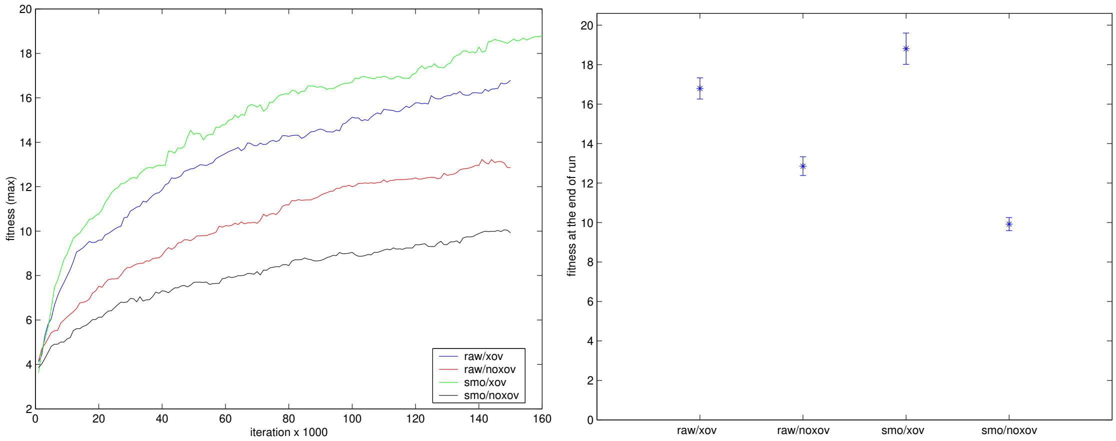


Figure 2.19: Comparison of two mutation strategies ("raw" and "smooth"), with or without use of crossover: Not using crossover results in a dramatic performance loss. Plots show maximum fitness reached (averaged over 10 runs) throughout the experiment (left) and at the end (right). Error bars show the standard error of the mean.

Results

To compare both types of mutation, we ran 10 iterations of the crane experiment (section 2.6.3) with four different mutation/crossover strategies.

1. Original (brittle) mutation + crossover
2. New (smooth) mutation + crossover
3. Original (brittle) mutation, no crossover
4. New (smooth) mutation, no crossover

The result was unexpected (fig. 2.19). We obtained slightly better performance with the improved mutation procedure. But the loss of performance when crossover is turned off was extremely large. As illustrated by the figure, after 150,000 iterations, the fitness reached, averaged over 10 runs, is 16.8 or 20.5 (depending on the mutation strategy) whereas without crossover the figures fall to 12.9 and 10, respectively.

This result suggests that recombination plays a key role, that of reusing useful subparts. Recombination, we posit, is responsible for the self similarity observed between different parts of evolved structures, and thus for the appearance of higher levels of organization than just the individual bricks.

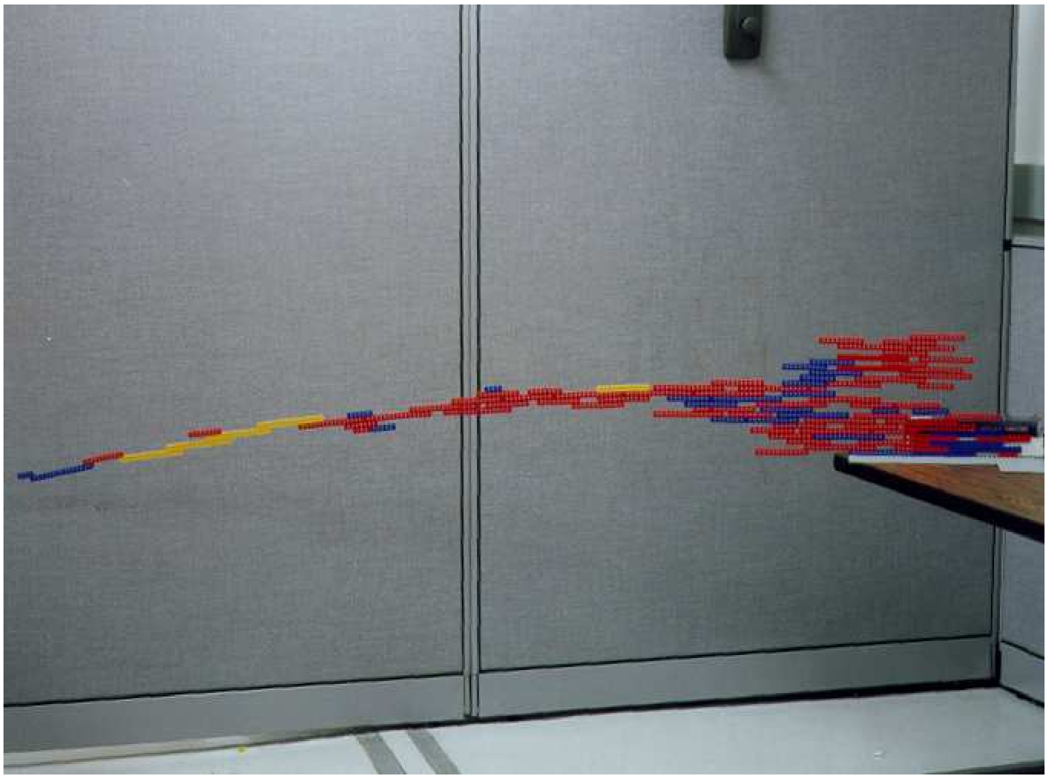


Figure 2.20: Long Bridge.

2.6 Artificial Evolution Re-Discovers Building Principles

In one experiment after another, our evolutionary setup for Lego structures came out with solutions that employ known principles of construction without really "knowing" about them. The recombination and mutation mechanisms are capable of finding construction principles due to the reuse of relevant subparts and their changes of roles.

2.6.1 The Long Bridge: Cantilevering

The "Long Bridge" experiment used an straightforward setup: A "base" of up to 40 knobs and a distant "target point" at (-300,0) (table 2.10).

We left the experiment run for a long time, until it ceased producing further improvements. The idea was to see how long a beam structure we could design. The resulting structure was larger than we had imagined was possible (figs. 2.20 and 2.21). The general principle discovered by this structure is that of cantilevering. A cantilevered beam is a well known architectural design problem, solved here by founding a thin, long, light beam on a strong base; counter-balancing it reduces stress (fig. 2.22).

Bricks	{4,6,8,10,12,16}
Max Bricks	127
Base	(0,-1)–(-39,-1)
x Range	(-310,41)
y Range	(-2, 80)
Initial Brick	6-brick at (0,0)
Target Point T	(-300,0)
Fitness(S)	$1 - \frac{d(S,T)}{d(0,T)}$

Table 2.10: Long bridge problem specification (x,y Lego units = 8mm $\times$ 9.6 mm.)

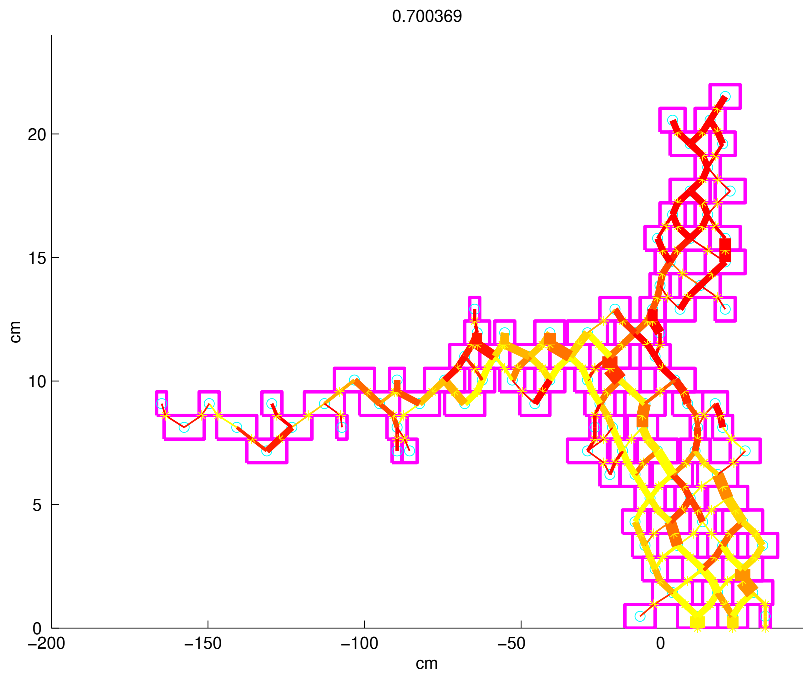


Figure 2.21: Long Bridge Scheme. The network represents the distribution of loads as assigned by our simulator; thicker lines correspond to stronger links several knobs wide. Light links are stressed to the limits, whereas dark ones are cool.

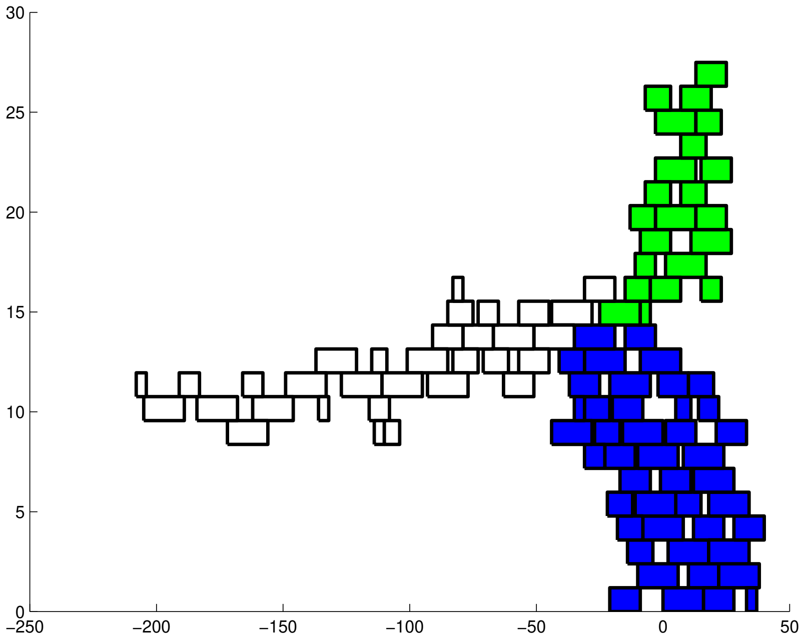


Figure 2.22: Long bridge organization: base (dark grey), cantilevered beam (white) and counterbalance (light grey).

Several hierarchical levels of organization are present in this structure:

- Level zero: Individual bricks.
- Level 1: Useful brick arrangements. Four-brick "boxes" (fig. 2.23) are an example.
- Level 2: Diagonal columns (the base is made up of layered columns).
- Level 3: Functional parts: base, beam, cantilever.
- Level 4: The entire structure.

Level 1 of complexity is interesting. Due to the widespread use of recombination, subsolutions such as the "brick box" are evolutionarily stable. They are used throughout the structure, be it beam, base or counterbalance, and give the structure a modular quality.

A random recombination has a higher chance of surviving if the bricks replaced are laid out in a similar pattern, thus evolutionary runs such as this one will favor recombinable structures.

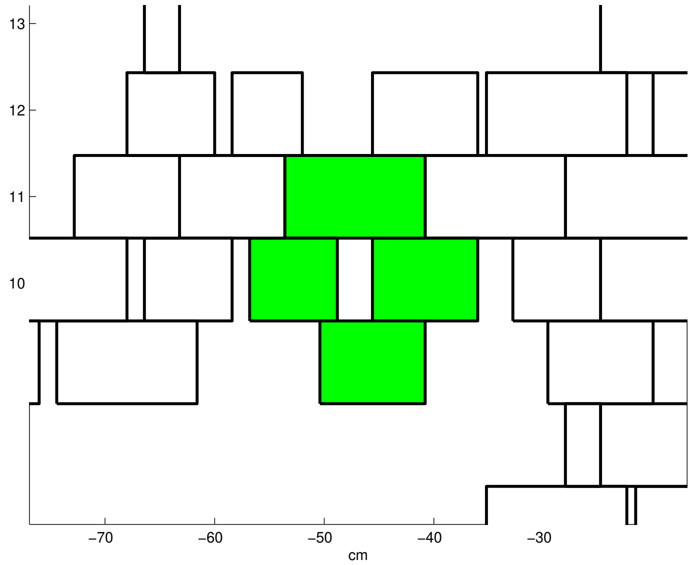


Figure 2.23: Four bricks in a “box” arrangement. This composite building block is used many times for the solution to the long bridge problem.

2.6.2 Exaptations, or ‘change of function’

Evolutionary biologists have coined the term *exaptation* to describe a commonplace occurrence in the evolution of life: a limb or organ evolves originally for a particular function, but later on is recruited to fulfill a new function [56]. When experimenting with our “embedded solver”, that encodes the simulation within the structure (section 2.2.7), we stumbled upon the role that this change of use fulfills in the long bridge experiment.

The idea of “encoding the simulation in the representation” is to save time by not letting the simulator search through the space of all possible paths on the support network. Instead, only one path is allowed, as encoded in the representation.

Figure 2.24 shows the detail from one run of the long bridge experiment using this new variant simulator. In fig. 2.21 every joint has a virtual axis of rotation (stars) which links two bricks by two edges. But with the compact simulator, only one direction is allowed, and consequently each joint has only one edge, and loads are allowed to flow in just one direction.

Fig. 2.24 is a close-up of an evolved bridge which has no ‘base’ part. The familiar cantilevered bar and counterbalance exist, but the base does not evolve. Why? Suppose a mutation or recombination creates a new brick at the point marked with X. A brick at such location would reinforce the base of the bridge, absorbing some of the weight of the cantilevered beam. This requires, however, a change of function of brick 2 above it. Currently, this brick rests over brick 1, balancing the beam. A base works in the opposite direction,

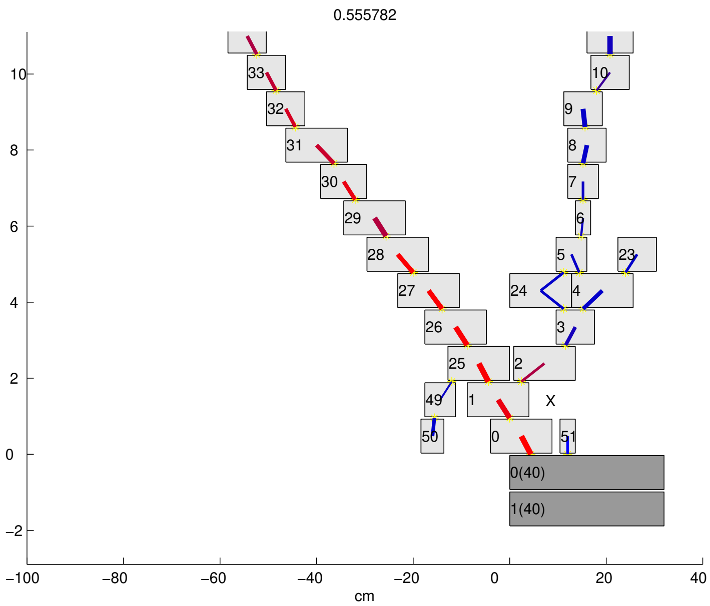


Figure 2.24: A run which encodes the simulation in the representation fails to change the use of counterbalancing bricks.

transferring load from 1 to 2 to X then to 5 and finally to the ground. The rest of the counterbalance would act at brick 2 instead of 1. With the simulation-in-the-representation scheme, this is unlikely to happen, for it requires the chaining of four mutations. In the meantime, the long beam cannot be supported and thus the entire structure collapses. After a long beam evolves that relies on a counterbalance, the change of function does not happen.

We conclude that in this case, the use of a realistic simulator allowed a change of use (from counterbalance to support) that cannot happen with a more limited simulator. Encoding the part together with its use resulted in an impoverished relationship between agent and reality.

2.6.3 The Crane: Triangle

The *diagonal crane arm* experiment had a slightly more complicated fitness function than the previous bridge experiment. Here the aim was to evolve a structure capable of holding a 250g payload up and away from a fixed base. The size of the external load is fixed but not its position (fig. 2.26).

We built by hand a crane base with motors for rotating the arm and pulling the hook. The evolved crane arm needed to attach to this base, so the experimental setup had 5 “fixed

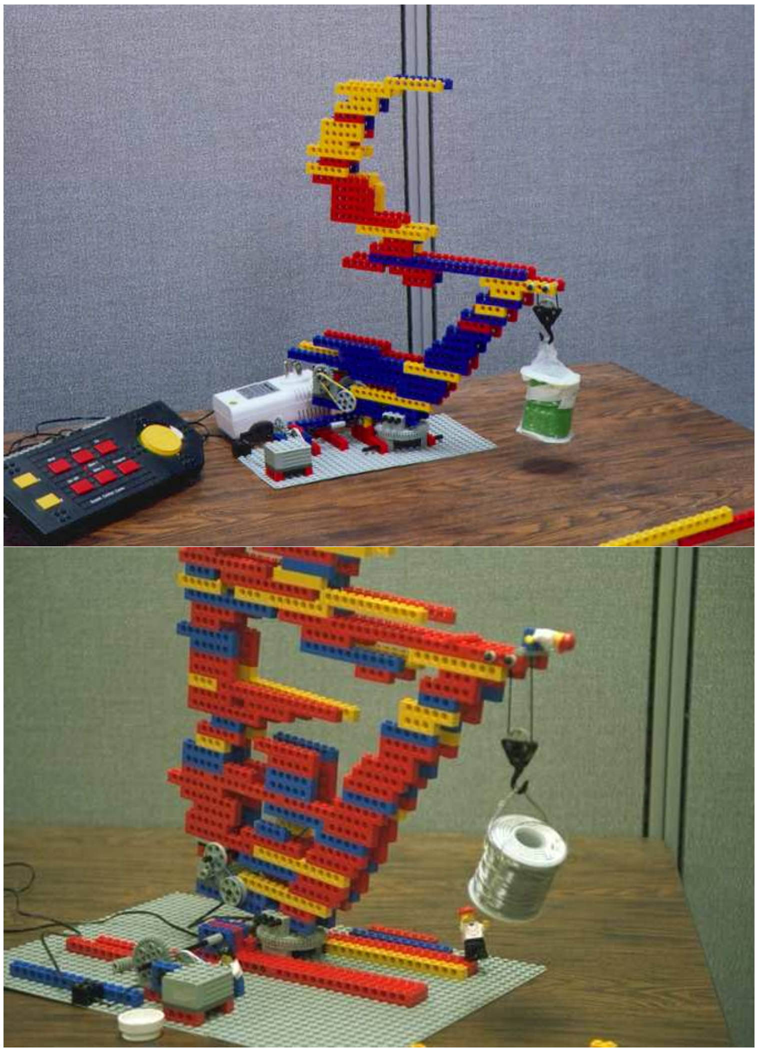


Figure 2.25: Crane with a diagonal crane arm: intermediate (top) and final (bottom) stages.

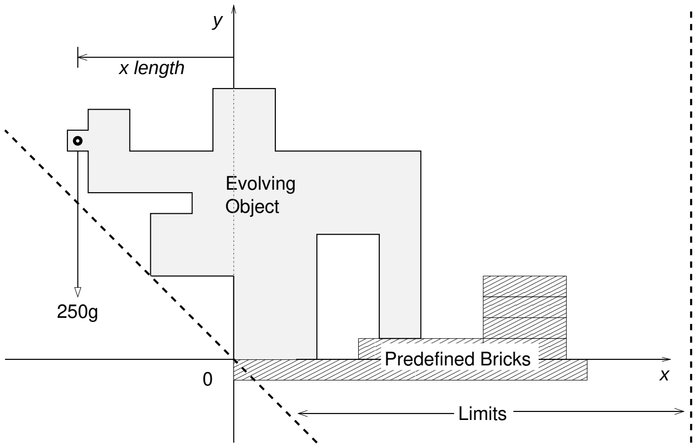


Figure 2.26: Crane arm experiment

bricks" where the rest of the structure was allowed to attach. The fitness value is the horizontal length x of the arm, but if the maximum load M supported at the tip is less than 250 g then x is multiplied by $M/250$, thus reducing the fitness (table 2.11). The arm has to go up and away at the same time, because the space is restricted to the diagonal semiplane $-x > y$.

We observed a curious phenomenon about the way a triangular solution appeared. Soon the crane evolved a counterbalance, located at the top, to act against the massive external load. This counterbalance took the shape of a bar going back from the tip of the crane — where the load is located, with a pile of bricks at the other end to act against the weight. This counter-weight could not be too heavy, for the crane is required to support its own weight before adding the external load (fig.2.27).

With the tendency to reuse useful parts that had already been observed in the long bridge experiment, the counterbalance at the tip, with its "J" shape, reappeared at a lower position, creating an arm counterbalanced at two points. The fact that these two counterbalances ended up connecting to each other and touching down at the base was a fortuitous event that created a new stronger synthesis: a triangular shape which supports more weight than the original counterbalancing bar, by tensioning the vertical column. At the same time the triangle supports all this counterbalancing apparatus (by compression) when the crane is not lifting a load. This is a change of use, as discussed in section 2.6.2, only in a much larger scale.

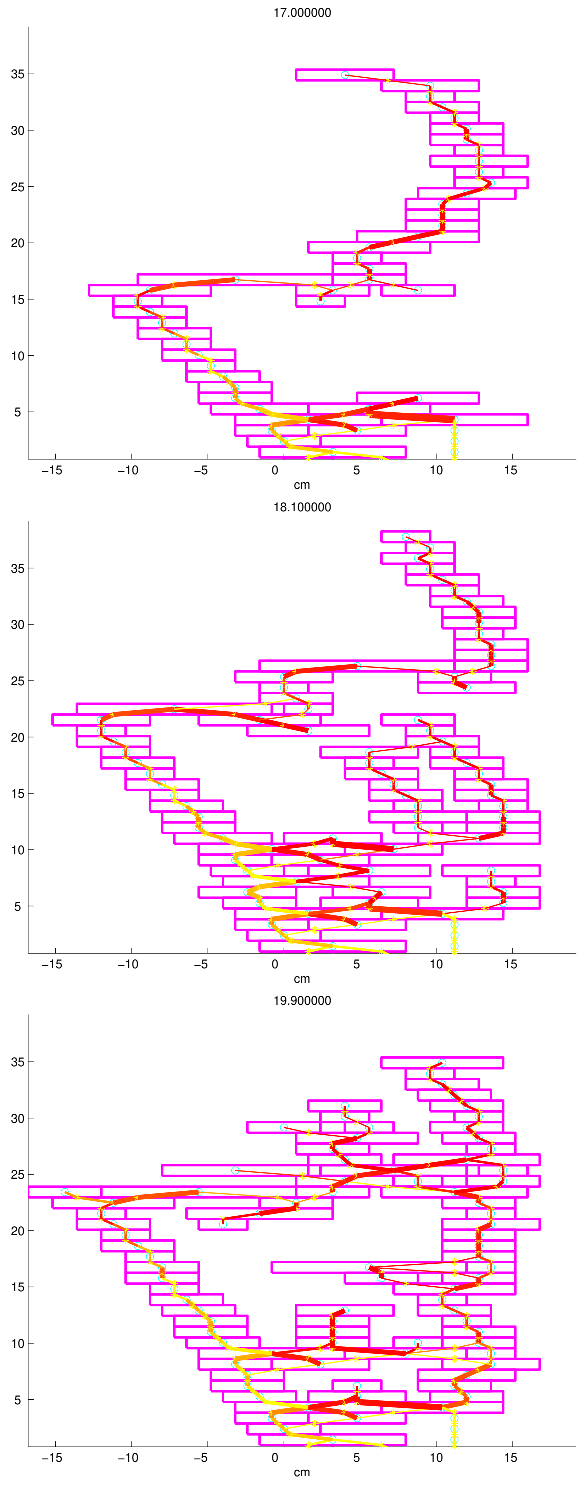


Figure 2.27: Three stages on the evolution of the diagonal crane arm: counterbalance, closer to triangle, closed triangle.

Bricks	{4,6,8,10,12,16}
Max Bricks	127
Base	(0,-1)–(-16,-1)
x Range	(-50,22)
y Range	(-1,40)
xy Restriction	y > -x
Fixed Bricks	6 at (6,0) 4 at (12,0) 4 at (12,1) 4 at (12,2) 4 at (12,3)
Fitness	$1 + (-x)\alpha$ x = position of the tip α = fraction of 250g supported

Table 2.11: Setup of the diagonal crane arm experiment.

2.7 Optimization

A comment that we often receive is that our final structures are not optimized: they contain redundant bricks that do not serve any apparent purpose. Of course, these irregularities are useful during the search process. Since we are not rewarding nor punishing for the number of bricks used, the evolutionary search freely generates variations with different numbers of bricks. All of them are potentially useful in the process of finding new combinations with higher fitness.

In a new run of the diagonal crane arm experiment, we added a little reward for lightness, inversely proportional to the number of bricks, but three orders of magnitude smaller than the raw fitness function. Fig. 2.28 shows two solutions for a crane arm the same length (a fitness value of 24). The structure on the right has a bigger premium, so we will prefer it.

Since we are willing to sacrifice everything else for the length of the arm, the fitness weight of the 'simplicity' factor has to be very small compared with the raw fitness measure (arm length). Among cranes of the same size and resistance, however, we prefer those with a smaller number of bricks. The evolutionary process is not biased against heavier versions of the crane; it just detects the simpler ones. In the example shown in fig. 2.28, fitness values of 24.0029 and 24.0040 have nearly identical chances of being selected in a fitness proportional selection scheme. But among otherwise identical cranes, the premium for implies that the cleanest one makes the final cut.

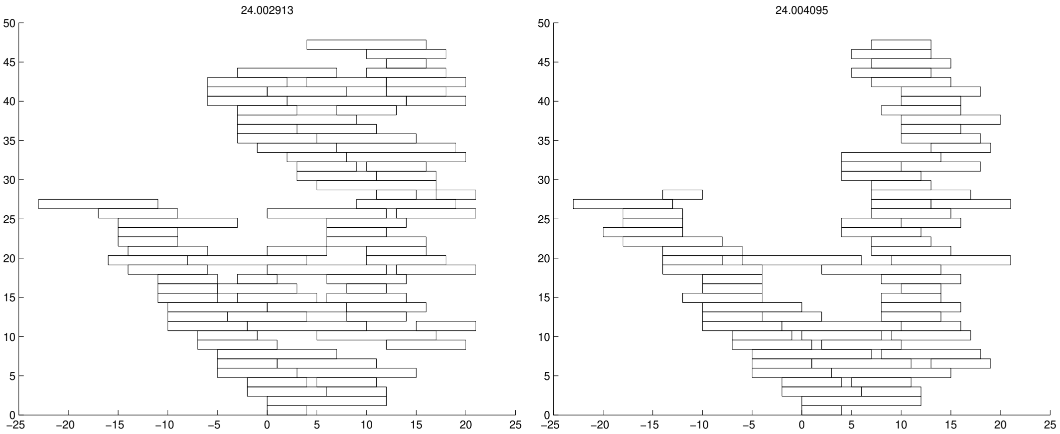


Figure 2.28: Optimization: Among several structures found with a raw fitness of 24, a small premium in the fitness function allows us to choose the one that uses less bricks (right). The tall vertical column (on the optimized version) cannot be eliminated because it acts as a counterbalance for the load that will be placed at the left tip of the crane.

2.8 Symmetry, Branching, Modularity: Lego Tree

The *tree* experiment was designed to test out whether some characteristics of natural trees (branching, symmetry) could evolve as a consequence of the environment. The design of a tree in nature is a product of conflicting objectives: maximizing the exposure to light while keeping internal stability.

The experimental design for the tree has a narrow attachment base: Only three knobs. This provides very little sustentation for cantilevering, so the structure will have to be balanced to reach out. A "light" resource, coming from directions up, left and right, has one value per column or row. Light is "absorbed" by the first brick it touches — and the fitness points given are equal to the distance from the absorption point to the x or y axis. The highest fitness would be a structure reaching out to completely cover the left, right and top borders (see fig. 2.29 and table 2.12).

There were no symmetry-oriented operators in our experiments, as could be, for example a "reverse" recombination operator that switched the orientation of a subpart. This means that symmetry is not encouraged by representational biases. Instead, the problem setup requires balancing the total weight of both sides. The tree did evolve, however, with a central symmetry with branches reaching out, by evolving the same type of solution separately on both sides.

The general layout of the evolved tree has several similarities with that of a real tree: there is a (somewhat twisted) trunk, with branches that become thinner as they reach out, and "leaves", bulky formations that maximize the surface at the end of the branch.

The tree is, among all our experiments, the one that most clearly illustrates the emer-

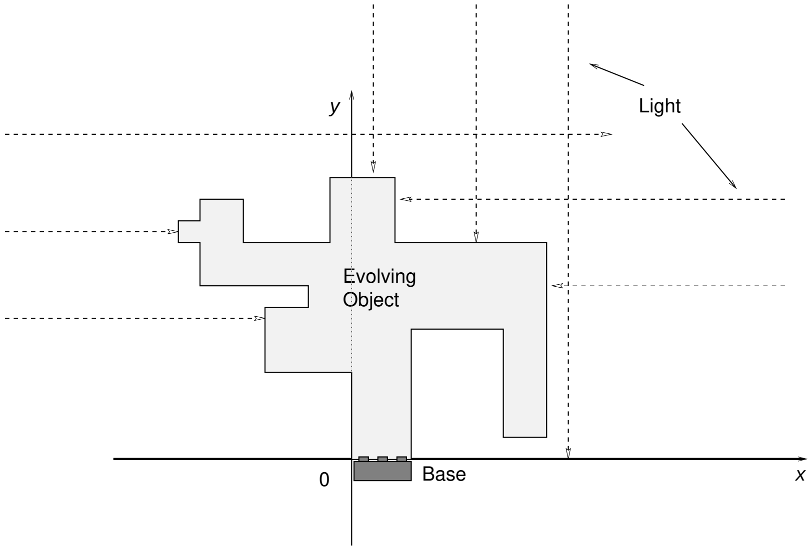


Figure 2.29: Tree experiment

Bricks	{1,2,4,6,8,10,12,16}
Max Bricks	127
Base	(0,-1)–(2,-1)
x Range	(-50,52)
y Range	(0,45)
Fitness	$f_L + f_R + f_T$ where $f_L = \sum_{j=0}^{45} \max\left(0, -\min\{x : (x, j) \in S\}\right)$ $f_R = \sum_{j=0}^{45} \max\left(0, \max\{x : (x, j) \in S\}\right)$ $f_T = \sum_{i=-50}^{52} \max\left(0, \max\{y : (i, y) \in S\}\right)$ S = structure

Table 2.12: Setup of the tree experiment.

gence of nested levels of organization, key indicator of what we call complex organization

- Level zero: individual bricks.
- Level 1: diagonal stacks and horizontal stacks of long bricks (on the branches), stacks of small bricks (at the tips),
- Level 2: trunk, U shaped branches, T structure at the top.
- Level 3: two tridents on top of each other, and a roof.
- Level 4: the entire structure.

2.8.1 Recombination Example

An interesting observation that illustrates the role of mutations is that the observed organization we have called "level 3" occurred as a result of one single crossover operation. In this lucky event (fig. 2.31), the bottom half part was reused to create also the top half part, discarding all the previous evolution of a top half that did not create a branching structure.

The "branches" organization proved to be evolutionarily more robust than the "fork" organization found initially, as it survived until the end of the run. The reuse of subparts and submodules is fundamental to generate these type of discrete transition.

2.9 Discovery is creativity: EvoCAD

Today's commercial CAD systems may add a mechanical simulator to the usual 3D manipulation tools[5]. But the new field of Evolutionary Design (ED) [11] has the potential to add a third leg to computer-aided design: A creative role. Not only designs can be drawn (as in CAD), or drawn and simulated (as in CAD+simulation), but also designed by the computer following guidelines given by the operator. Thus we envision future Evolutionary CAD systems, "EvoCADs."

An EvoCAD system has the human designer in the main role: the designer has an idea or concept for a required object. Some of the requirements can be added to the 3D canvas, creating evolutionary targets that an ED engine uses for evolving a possible design. The output of this evolutionary engine can be modified, tested and re-evolved as many times as desired (figure 2.32).

To demonstrate our conception of EvoCAD, we have built a mini-CAD system to design 2D Lego structures. This Lego EvoCAD allows the user to manipulate Lego structures, and

[5] PTC's Pro/Engineer software, whose CAD tool can generate output for the mechanical simulator, Pro/Mechanica, is an example.

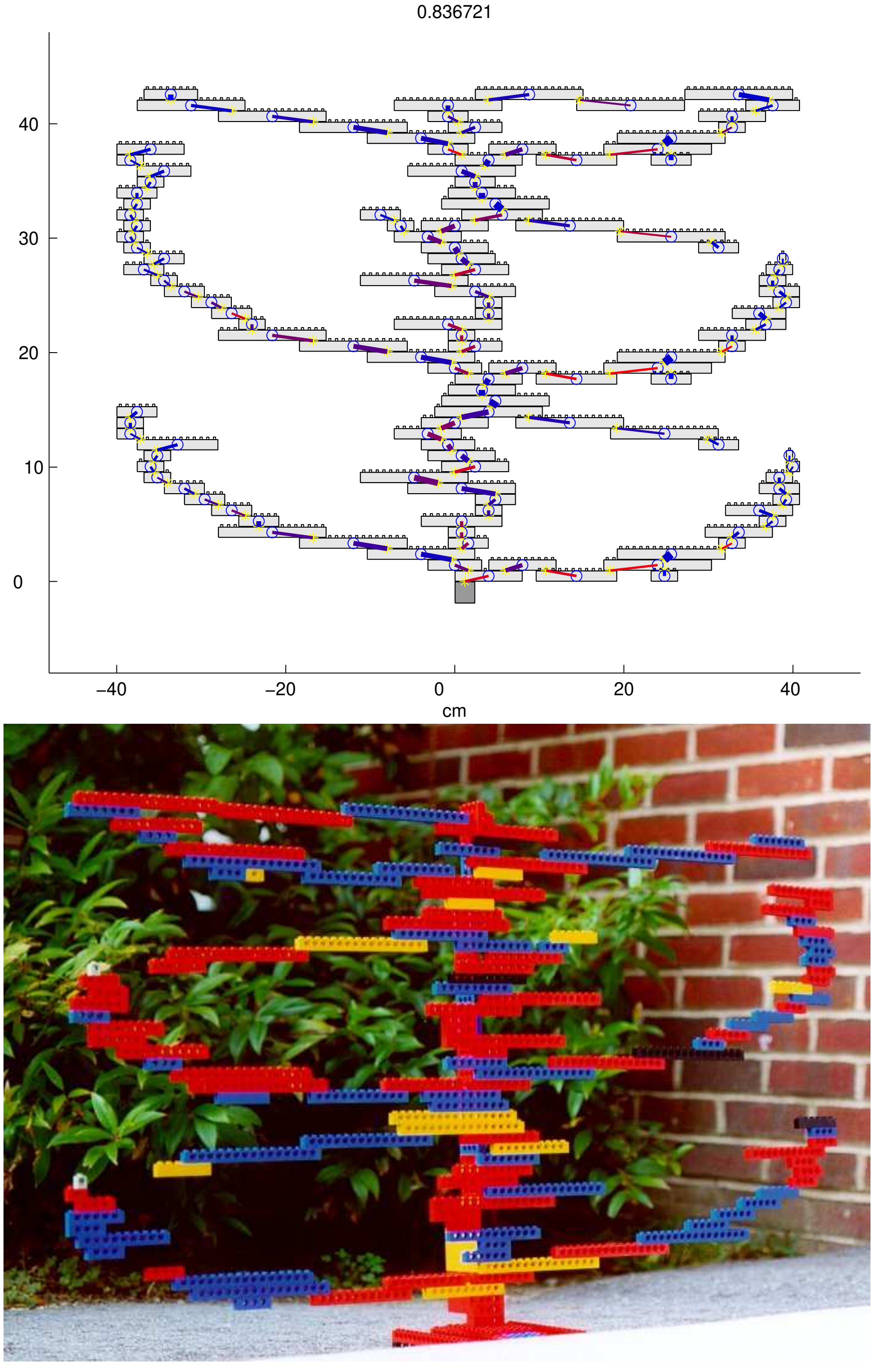


Figure 2.30: Evolved Tree (internal model and built structure)

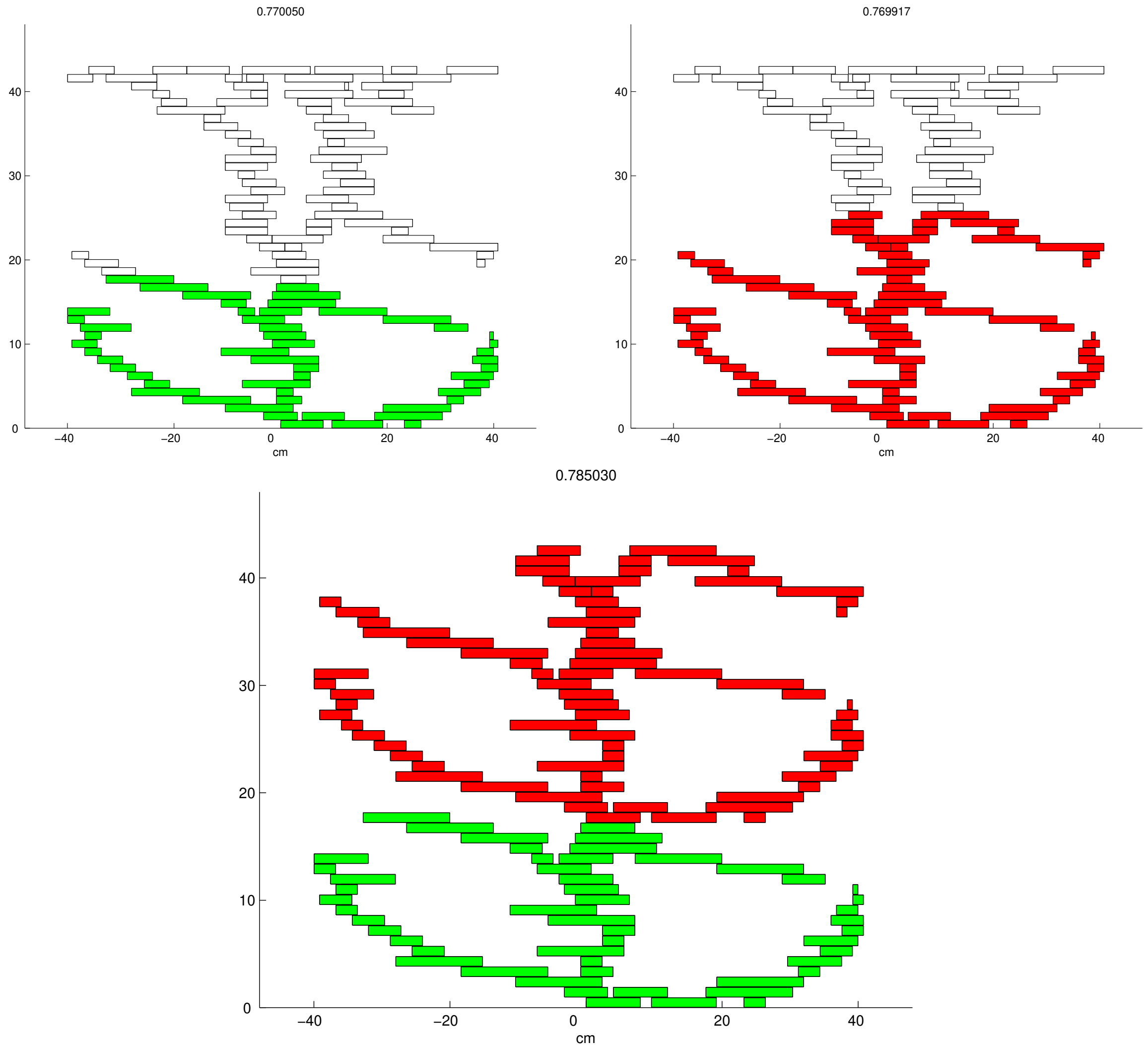


Figure 2.31: *Recombination*. An interesting example of a recombination at the highest level, that creates a recursive structure. The two parents (top) are close relatives, very similar. The root parent (top left) got the crossover point in the trunk of the structure; bricks colored white were discarded by the operation. The secondary parent (top right) had its insertion point at the root. The resulting structure is shown at the bottom. Besides the bricks lost by the root parent's replacement of a subtree, other bricks were lost during the development process: the secondary parent lost all its top bricks (colored white) because they became out-of-bounds, and the central-left branch of the root parent lost three bricks because the maximum bricks limit (127) was reached. The final version of the tree (fig. 2.30) evolved from this individual.

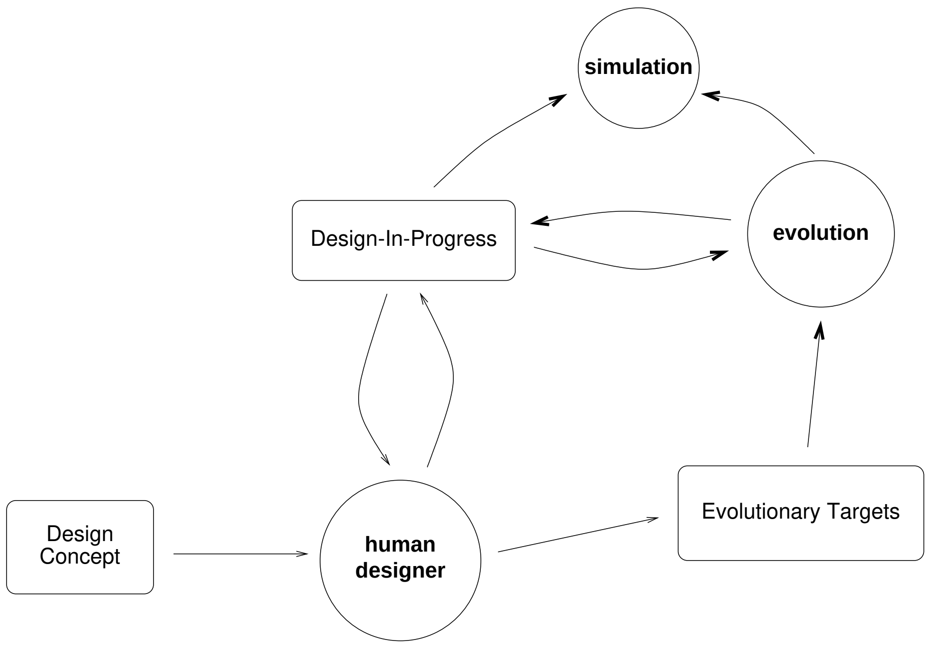


Figure 2.32: A conceptual EvoCAD system has two "creative minds", the human designer and the ED software. The human designer is in control, and calls upon the remaining elements as tools. A problem description language (PDL) allows CAD, evolutionary and simulation components to communicate with each other (bold arrows).

test their gravitational resistance using the same structural simulator we have been using to do ED with Lego bricks. It also interfaces to an evolutionary algorithm that combines user-defined goals with simulation to evolve candidate solutions for the design problems. The results of evolution are sent back to the CAD front-end to allow for further re-design until a satisfactory solution is obtained.

2.9.1 Evolutionary Algorithm

Instead of initializing the population with a single brick, as in previous experiments, here we want the current design to be used as a starting point.

To begin an evolutionary run, a starting structure is received by the ED engine, consisting of one or more bricks, and needs to be "reverse-compiled" into a genetic representation in order to seed the population.

Mutation and crossover operators are applied iteratively to grow and evolve a population of structures. The simulator is run on each new structure to test for stability and load support, and compute a fitness value. Evolution stops when all objectives are satisfied or when a timeout occurs.

Object type	Parameters	Comments
brick	x, y, size	Bricks will compose the initial seed of the evolving population.
fixed brick	x, y, size	Bricks that are not modifiable by evolution.
ground	x, y, size	At least one needed. One or more grounds support the structure.
load	x, y, magnitude	Loads that must be supported by the structure.
target point	x, y	Points that must be touched by the structure.
restriction	x, y	Points that are unusable.
canvas limits	min x, max x, min y, max y	Describes canvas size to evolutionary engine, establishing boundaries for the simulator.

Table 2.13: Brick problem description language (BPDL) symbols are used to send problems and solutions between evolving engine, CAD screen and simulator

2.9.2 Brick Problem Description Language

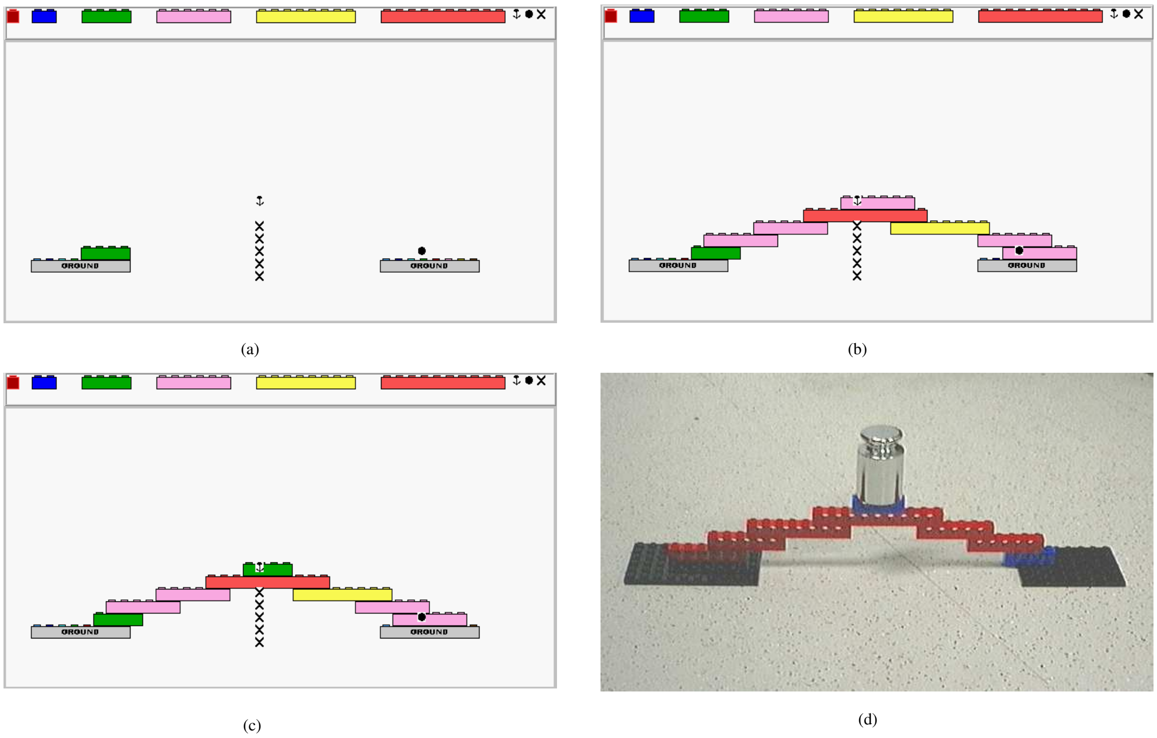
We designed a Brick Problem Description Language (BPDL), as an interface between the evolutionary algorithm, simulator, and the CAD front-end. When the user clicks the "evolve" button, a BPDL description is sent over the Internet to an evolution server which evolves a solution for the problem. The result of the evolution is sent back to the CAD using the same language. The simulator receives BPDL-encoded structures for testing, both from the CAD (when the human wants to test a structure) and from the evolutionary engine, which tests every mutated or recombined structure .

2.9.3 Target Points and Target Loads

The goals for the ED engine are deduced from user-defined *Restrictions*, *Target Points* and *Target Loads*. A structure will be fully satisfactory if it touches all the target points and target load points, whereas avoiding all the restricted points, and supports all the specified loads at each target load point.

2.9.4 Fitness Function

All previous experiments used handwritten fitness functions to guide the search For Evo-CAD we need instead to compute a generic fitness value for any BPDL structure. Although optimized fitness functions will require further study, the fitness of a structure S has been



(a) (b)

(c) (d)

Figure 2.33: Sample working session with the EvoCAD program: (a) The user has defined two grounds and several evolutionary hints: restrictions (x), target (dot) and load (arrow). An initial brick was laid down. (b) Evolution designed a structure that fulfills all requirements. (c) The user made cosmetic corrections (d) The structure has been built with Lego bricks.

defined as:

$$\sum_{t\in\text{targets}} \frac{1}{1+d(S,t)} + \sum_{l\in\text{loads}} \frac{1}{1+d(S,l)} + \sum_{\substack{l\,\in\,\text{loads}\\ d(S,l)=0}} \text{supp}(S,l) \tag{2.13}$$

(where d computes the distance between a point and the nearest brick in the structure, and *supp* uses the simulator to compute the fraction of a certain load that the structure supports)

2.9.5 Results

Our Lego EvoCAD system (figure 2.33) demonstrates how a new kind of application could employ ED techniques to let the computer not only be a canvas and a simulation tool, but also create its own designs following the users' specifications. Our system allows human and computer to create a design collaboratively, greatly reducing the human effort needed to craft and optimize a design.

2.10 Discussion

2.10.1 Modeling and the Reality Gap

The properties of bricks are variable. Differences in construction, age, dirt, temperature, humidity and other unpredictable factors produce seemingly random variations when their behavior is measured. These factors had to be considered in order to have buildable results. We have accounted for this problem using a safety margin: our model assigns 20% less resistance to all the joints involved.

The simplistic simulator described is far from modeling physics to its full detail, yet any model for modular structures, no matter how accurate, has to compensate for the random variability in the generic properties of the average brick, using similar methods. The value of 20% was set intuitively and may require further study, especially as our structures scale up in size and complexity.

Engineers have been using safety margins for a long time, to deal with all the factors that cannot be calculated ahead of time. In ALife, evolutionary roboticists have found unpredictabilities when attempting to simulate the environment for a real robot [72, 98]. One simple technique used in ER is to add random noise to the simulated sensors in order to generate robust behaviors suitable to be transferred to the real world.

Noise and safety margins are variations on a fundamental principle of conservativeness. The so-called "reality gap", which is the difference between the model and the final artifact, does not mean that modeling is useless. But it must be taken into account in order to achieve successful transfer from simulation to reality. ALife research has addressed the reality gap problem in different ways, such as:

- Using safety margins, as exemplified by the present work.
- Evolving directly in reality, as in evolution in FPGA's [133] or robot behaviors [37].
- Evolving adaptable entities, as proposed by Di Paolo [32].
- Use a hybrid model, first evolving in simulation, then transfer to reality to do a final round of adaptation, has been proposed by our group [110] and implemented (for a game) in the two-layer evolutionary architecture of Tron (chapter 3).

We agree with Jakobi [70, 71] in going for simulation at the right level, aiming at the aspects of reality that insure robust, transferable behaviors. We also think that the method of simulated evolution has inherent advantages in lieu of the reality gap problem: the dynamics of reproduction and selection subject to mutation lead evolving populations to explore fuzzy regions — clouds within the solution space.

Those organisms that, when mutated, are likely to be successful, have a reproductive advantage: their offspring is more likely to survive. This means that inasmuch as genotypical and phenotypical variation are linked, evolved solutions will have a tendency to being robust, in the sense that small variations in their constitution will not incapacitate them (more likely, they give rise to successful mutants). If this phenomenon is true, then solutions obtained by evolution implement "by default" a conservative principle based on kinship. This is an open research question.

2.10.2 Modular and Reconfigurable Robotics

In our 1998 article [48] we proposed that this type of technique should be applied in ER to evolve the body and brain of a robot simultaneously. We thought that the modular approach should fit well with the research in modular robots, such as proposed by Yim [139] or Pamecha [104].

However, the first realization of fully coevolved body and behavior came two years later with Lipson and Pollack's crawlers [94]. Instead of a modular parts approach, they use a continuously deformable architecture, employing a rapid-prototyping machine to build the evolved components on-the-fly (fig. 2.34).

2.10.3 Movement Planning as Evolution?

Evolutionary algorithms solve a problem by maintaining a population of variant partial solutions and applying recombination operators to them. Those operators can be considered

Figure 2.34: Walking robot evolved by Lipson and Pollack builds further on our paradigm of evolution of structure under buildability constraints, adding movement and control.

valid, or available operations in a space of configurations. Our brick structures algorithms are solving problems of spatial arrangement, subject to buildability restrictions, starting from a known initial configuration and advancing, one step at a time, by legal transformations. The simulation enforces that each step is physically correct.

The implication is that the succession of genetic transformations that yield to a final stage can be considered a plan. Problems of planning for spatial arrangement are classic in AI [35]. One plausible future application of structural evolutionary algorithms is to adapt the recombination operations to reflect valid reconfigurations of a modular metamorphic robot. Problems of robot locomotion for example could be solved by evolving a plan for the desired final configuration from the starting one.

We ran an exploratory experiment which evolves an imaginary amoeba-like creature made of Lego bricks (fig. 2.35). Once a goal state is obtained, the "family tree" from the initial to final configurations represents a sequence of valid transformations.

The mutation operations are interpreted as the "valid actions" of the creature (i.e., extending, retracting or displacing limbs) and recombinations amount to macro operators, chaining a sequence of actions.

2.10.4 Cellular and Modularity-Sensitive Representations

The problem of neural network representations for evolution of recurrent networks is similar to our problem of encoding brick structures. From early naive representations the

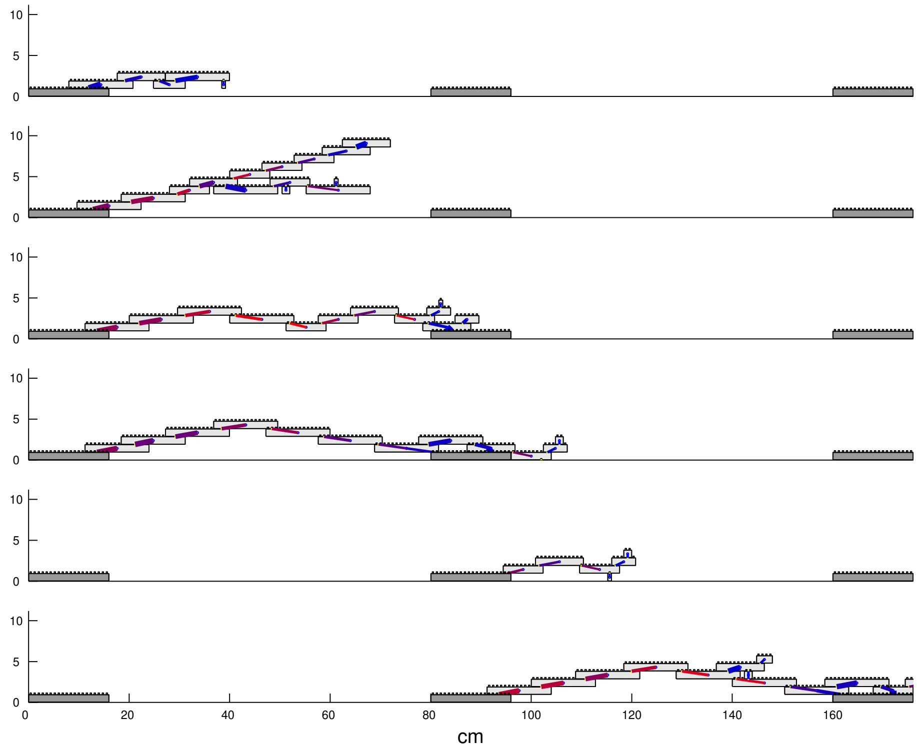


Figure 2.35: An imaginary Lego creature evolves a plan to locomote between islands by expanding and contracting its body in “amoeba” fashion. Each step involves adding, deleting or sliding a limb, as induced by mutations. Recombination works as macro operator.

concept of ‘developmental’ or ‘cellular’ grammatical encodings emerged [9, 60, 78]. They increase the efficiency of the GA by reducing the search space, eliminating redundancies and meaningless codes, and providing meaningful recombination operators.

There is a developmental stage in our experiments, because the genotype builds a phenotype by laying down bricks one at at time, and fails whenever the position indicated is invalid, either because a previous brick is occupying that position already, is out of bounds, or the maximum numbers of bricks has been reached (see eqs. 2.10, 2.11 and fig. 2.31). Each failed brick results in the deletion of a subtree.

An interesting alternative, never tested, would have been to delete illegal bricks from the phenotype but not the genotype, thus allowing for ghost limbs that could reappear later on. In any case, our representation has no means to represent subroutines or iterations other than interchanging genetic material via recombination.

There have been studies of modularity in recurrent neural net representations [5, 61], aiming to improve the GA with automatic creation of libraries of reusable subcomponents. Structural representations should ultimately aim at the same objective: Finding useful complex blocks and incorporating them in the making of large structures with inherent modularities. Recent work by Hornby, et. al [66] utilizes an L-system generative representation to evolve walking virtual creatures.

2.10.5 Surprise in Design

Instead of devising an expert system with rules about how to divide a task into subtasks, and how to carry along with each of those, a system like ours relies on more basic assumptions. The rules of physics, unlike the rules of design, are not an artificial creation, and this leads to original designs, because the evolutionary algorithm explores *design space* in ways which are not so strongly pre-determined by our culture.

This is the reason why our evolved structures have an alien look, which prompted Ronald and Sipper to observe that they exemplify “surprising surprise, where we are totally and utterly taken aback” [116, p. 525]. According to them, these Lego artifacts lack the familiar look that helps us trust and use a structure produced by classic engineering. However, we see their comments as a strong encouragement, for among the most ambitious goals of AI is that of expanding the human mind by introducing new ways of thinking. We believe that useful inventions, no matter how weird they might look in the beginning, are eventually incorporated into the culture if they are useful. Just as today we trust our lives to an airplane (which at first glance seems utterly incapable of flight), tomorrow we may walk over bridges designed by ALife.

2.10.6 Conclusions

In machine learning and artificial evolution systems, the more interesting results, such as Sims' creatures or expert Backgammon players, are due more to elements of the learning environment than to any sophistication in the learning algorithm itself [108, 132]. By keeping inductive biases and *ad hoc* ingredients to a minimum, we have demonstrated that interesting real-world behavior can come from a simple virtual model of physics and a basic adaptive algorithm.

The use of modular building elements with predictable — within an error margin — properties allows evolutionary algorithms to manipulate physical entities in simulation in ways similar to what we have seen, for example, in the case of robot control software. The bits in our artificial chromosomes are not limited to codifying just bits; they are capable of representing the building blocks of an entire physical structure.

We believe to have only scratched the surface of what is achievable. Combined with suitable simulators, the recombination of modular components guided by an artificial selection algorithm is a powerful framework capable of designing complete architectures ready to be built and used, discovering and exploiting complex properties of the substrate which are not identical to those explored by human engineers and designers.

Chapter 3

Coevolving Behavior with Live Creatures

3.1 Introduction

In the 1966 movie classic *Fantastic Voyage* [1], a group of humans were shrunk and inserted inside a human body to cure a disease. Sixteen years later, *Tron* [137] shrunk film heroes into a new fantastic domain: *the world inside the computer.* This new pop icon sprang up from the massive impact that the 8-bit microprocessor had in our culture, bringing us personal computers and an explosion of arcade video games.

Virtual worlds as studied by Artificial Life are inhabited by creatures that reproduce, learn and evolve — with varying degrees of physical realism — without human participation. For the virtual worlds of video games, instead, humans are invited but the emphasis is on the visual appeal: artificial opponents are present but they rely on access to the internal variables of the game more than artificial adaptation — they are not artificial life beings.

Robots interact with physical reality and live creatures: they are *embodied.* But robotics research is difficult because the real world brings with it limitations of space, budget, engineering and speed. A video-game instead, is a simulated world where human intelligence can meet artificial adaptation, through the *immersive* experience of virtual reality.

Here we posit that the science of Artificial Life should employ the metaphors of virtual realities and video games to attain knowledge about adaptive behavior, putting artificial agents in contact with live creatures, by introducing them into a simulated world. Virtual worlds could enable ALife researchers to study how artificial and natural intelligence coevolve, adapting to each other. Moreover, with the revolution of the Internet, these worlds can reach thousands of human subjects and artificial learning can arise from the combined contributions of many individuals.

We present the first work that evolves agents by having them play against humans. The recreational nature of a simple video game attracts people and creates a niche for mutual adaptation between people and agents, providing the substrate for the first experience in learning a game through the massive training by thousands of human subjects.

The physical features of our model are limited to the simulation of a two-dimensional space with walls; but they are sufficient to observe the emergence of navigational behaviors such as wall following and obstacle avoidance.

Parts of this research have been reported on the following publications: [44,50,51,125].

3.2 Background and Related Work

3.2.1 Coevolution

In nature, organisms and species coexist in an ecosystem; each species has its own place or *niche* in the system. The environment contains a limited number and amount of resources, and the various species must compete for access to those resources. Through these interactions, species grow and change, each influencing the others' evolutionary development. This process of reciprocal adaptation is known as coevolution.

In evolutionary computation, the term "coevolution" has been used to describe any iterated adaptation involving "arms races", either between learning species or between a learner and its learning environment. Examples of coevolutionary learning include the pioneering work by Hillis on sorting networks [65], Backgammon learning [107, 108, 131], predator/prey games [25, 101, 114] and spatial distribution problems [74, 75]. The present work extends the coevolution paradigm to include the case where the changing environment results from the adaptive behavior of a heterogeneous population of human beings.

3.2.2 Too Many Fitness Evaluations

The need to evaluate the fitness of a large number of individuals is a critical factor that restricts the range of application of GA's. In many domains, a computer can do these evaluations very fast; but in others, the time spent by this process may render the GA solution impractical. Examples of the latter case include a computer playing a game with people and trying to learn from experience or a robot attempting to complete a task in the physical world.

Robots that are reliable enough can run repeated trials of the same experiment over a long time in order to learn using evolutionary computation techniques. Floreano and Mondada [37, 38] run their robots for several days in order to evolve controllers for basic tasks. Most evolutionary roboticists have preferred to rely on computer simulations to provide them with faster evaluations, but the crafting of appropriate simulators is also very difficult [98].

Evolution through interaction with humans faces similar difficulties. “Blind watchmaker” systems, where the user ranks every generation manually, have been successfully used to evolve shapes [30, 86]; even with the extreme limitations imposed by the need to evaluate each individual manually (minimal population sizes, small number of generations) those results prove the great potential of evolution in a human-generated landscape.

But with a human in the loop, it is impossible to attain the large numbers of generations and evaluations employed in evolutionary experiments. Humans — unlike robots — get tired of repetitive tasks. Moreover, humans act irregularly; they may react differently each time when faced with the same situation more than once. If users provide fitness evaluations, adaptive software would need to be able to filter out such sources of “noise” provided naturally by human users.

We believe that the Internet, with millions of human users, could be fertile ground for the evolution of interactive adaptive software. Instead of relying on a few selected testers, the whole community of users together constitutes a viable gauge of fitness for an evolutionary algorithm that is searching to optimize its behavior.

3.2.3 Learning to Play Games

Self-Play or Play against People?

A machine that learns by playing games may acquire knowledge either from external expertise (playing with a human or human-programmed trainer), or by engaging in *self-play*.

Tesauro [131] was able to obtain strong Backgammon players, having one neural network play itself and adjusting the weights with a variant of Sutton’s TD algorithm [128]. Although it worked for Backgammon, self-play has failed on other domains. Our group obtained similar results to those of Tesauro using hill-climbing, a much simpler algorithm [109]. This demonstrated that elements unique to Backgammon, more than the TD method, enable learning to succeed. Self-play remains an attractive idea because no external experience is required. In most cases, however, the learning agent explores a narrow portion of the problem domain and fails to generalize to the game as humans perceive it.

Attaining knowledge from human experience has proven to be difficult as well. Today’s algorithms would require millions of games, hence rendering training against a live human impossible in practice. Programmed trainers have also led to the exploration of an insufficient subset of the game space: Tesauro [130] tried to learn Backgammon using human knowledge through a database of human expert examples, but self-play yielded better results. Angeline and Pollack [4] showed how a genetic program that learned to play tic-tac-toe against several fixed heuristic players was outperformed by the winner in a self-playing population. Most of today’s expert computer players are programmed by humans; some employ no learning at all [103] and some use it during a final stage to fine-tune a few internal parameters [7]. A recent exception is Fogel’s checkers player [40], which achieved a "Class A" rating by coevolutionary self-play alone.

Real-time, interactive games (e.g. video games) have distinctive features that differentiate them from board games. Koza [84] and others [117] evolved players for the game of Pacman. There has been important research in pursuer-evader games [25, 101, 114] as well as contests in simulated physics environments [123]. But these games do not have human participants, as their environments are either provided by the game itself, or emerge from coevolutionary interactions inside a population of agents.

3.2.4 Intelligence on the Web

A space where agents can thrive and evolve

It has been suggested that intelligence should be present on Internet sites in the form of intelligent agents. According to this hypothesis, such environments will contain software agents that interact with human users and adapt according to the behavior displayed in those interactions [93].

With Tron we are exploring the hypothesis that one of the forms in which this idea may be realized is through the presence of species of agents evolving through their interactions with the rest of the web. From this perspective, the Internet is seen as a complex environment with virtual niches inhabited by adaptive agents.

Here we propose that learning complex behaviors can be achieved in a coevolutionary environment where one population consists of the human users of an interactive software tool and the "opposing" population is artificial, generated by a coevolutionary learning engine. A niche must be created in order for the arms race phenomenon to take place, requiring that:

1. A sufficiently large number of potential human users must exist.

2. The artificial population must provide a useful environment for the human users, even when — in the early stages — many instances perform poorly.

3. An evaluation of the performance of the artificial population must be measurable from its interaction with the human users.

The experimental learning environment we created for the game Tron met these requirements. First, the game is played in a Java applet window on our web site. As Tron was being launched, Java was a new thing and there was a great interest on any applications, particularly games. So by advertising our site in Java games lists we were able to attract visitors. Second, our earlier experiments with Tron had shown us that, by self-play, we could produce players that were not entirely uninteresting when faced by humans. And third, each round of Tron results in a performance measure: a win, a loss or (rarely) a tie.

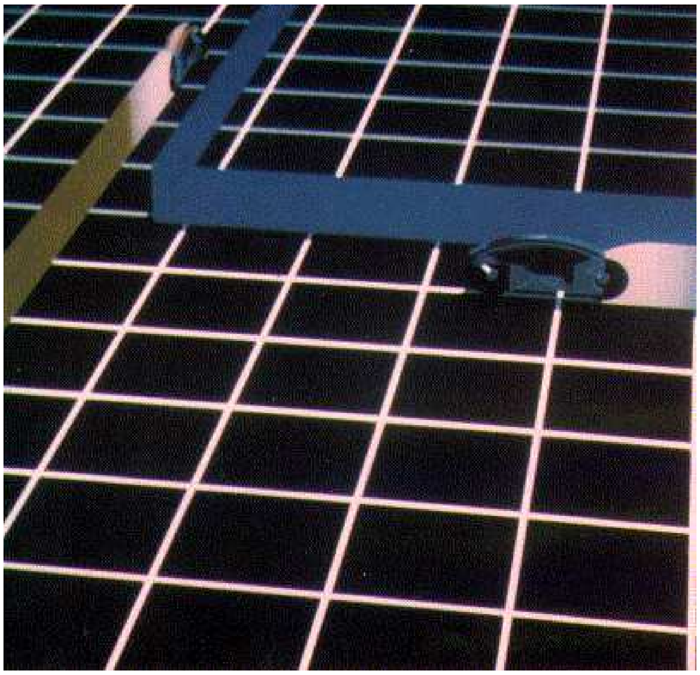


Figure 3.1: *Light Cycles.* Still from the movie *Tron.*

3.3 Experimental Model

3.3.1 Tron Light Cycles

Tron is a popular video game that has been implemented in arcades and PC's with different rule variations. It is based upon a segment of the movie *Tron* [137], where the characters rode *Light Cycles*, futuristic motorcycles that leave a solid trail behind them (fig. 3.1).

The Tron-Light Cycles game is a contest between opponents who move at constant, identical speeds, erecting walls wherever they pass and turning only at right angles. As the game advances, the 2D game arena progressively fills with walls and eventually one opponent crashes, losing the game.

Tron requires quick reactions and spatial-topological reasoning at the same time. In our version, the two players (one human, one agent) start in the middle region of the screen, moving in the same direction. The edges are not considered "walls"; players move past them and reappear on the opposite side, thus creating a toroidal game arena. The size of the arena is 256×256 pixels.

Fig. 3.2 shows the Java version of our Tron game, running inside an Internet browser application. A Java agent is constrained by the Java Virtual Machine of the browser, an environment very limited in speed and resources. At the time when our experiment was conceived, Java was a new technology and only small, simple programs would run reliably on commercial web browsers. The minimalistic memory, CPU and graphics requirements of Tron made it an ideal choice for the experiment.

Initial Experiment

In exploratory experiments [43], we used a Genetic Algorithm to learn the weights of a perceptron network that played Tron. We found that simple agents played interestingly, but also that coevolution may not always lead to robust strategies. *Collusion* [107] was likely to appear in the form of "live and let live" strategies such as the one shown in figure 3.3, where agents avoid confrontation and "agree" to tie, splitting the point.

3.3.2 System Architecture

Tron is implemented as a client/sever application with three main components (fig. 3.4),

- *Java Front End*. A Java applet that runs on web browsers, playing Tron between a human and an agent.
- *Main (Foreground) Server.* An Internet web server and SQL database that hosts a population of Tron agents, evolving it against humanity. This server records all games played, computes fitness and decides which agents live and die.
- *Novelty (Background) Engine.* An application that evolves Tron agents by self-play. It sends new agents to the Foreground Server whenever they are needed, and receives veteran champions to be used as fitness measures and/or seeds for a new population.

3.3.3 Tron Agents

Tron agents perceive the world through sensors that evaluate the distance in pixels from the current position to the nearest obstacle in eight relative directions: front, back, left, right, front left, front right, back left and back right. Every sensor returns a maximum value of 1 for an immediate obstacle, a lower number for an obstacle further away, and 0 when there are no walls in sight (figs. 3.5 and 3.2).

Each agent or "robot" is a small program, representing one Tron strategy, coded as a Genetic Programming (GP) s-expression [84], with terminals {A, B, ..., H (the eight sensors) and $\Re$ (random constants between 0 and 1)}, functions {+, -, * (arithmetic operations),% (safe division), IFLTE (if less or equal-then-else), RIGHT (turn right) and LEFT (turn left)}, maximum depth of 7 and maximum size of 512 tokens. An agent reads its

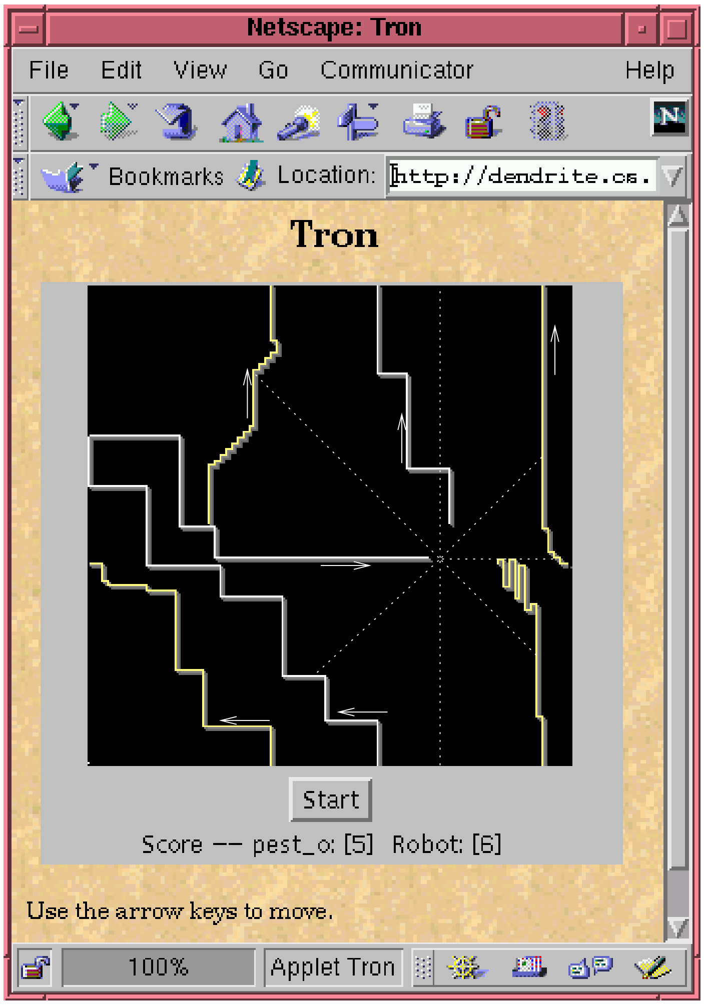


Figure 3.2: The Tron page: Tron runs as an applet inside an Internet browser. Arrows have been added to indicate direction of movement, and dotted lines to show the sensors of an artificial agent.

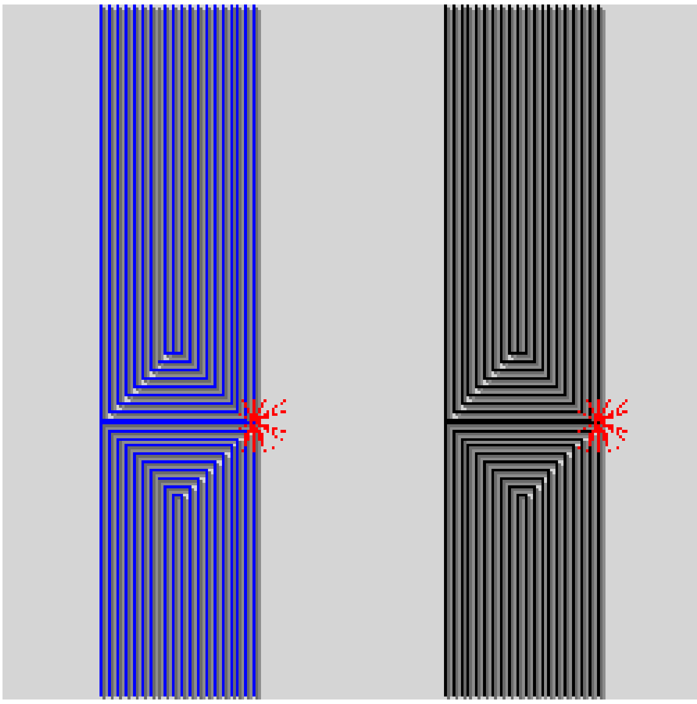


Figure 3.3: Live and let live: Two artificial Tron players make tight spirals in order to stay as far from the opponent as possible. This form of collusion is a frequent suboptimal equilibrium that complicates artificial learning through self-play.

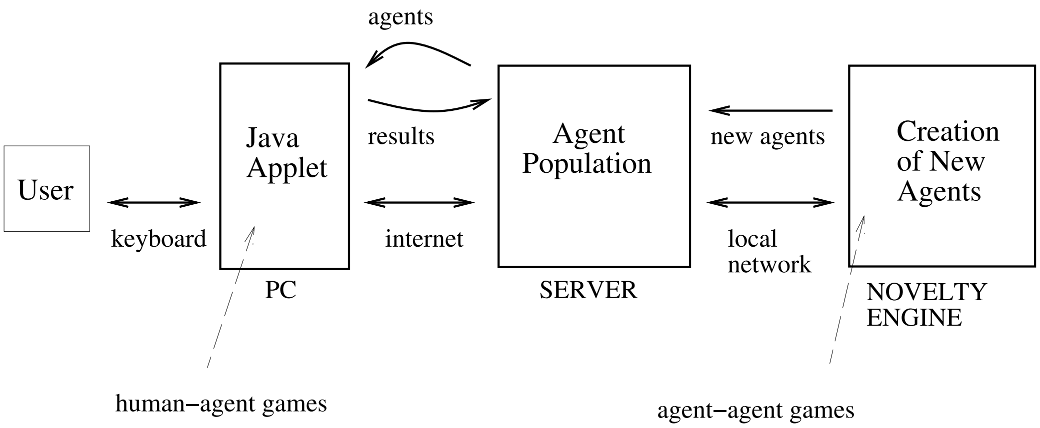


Figure 3.4: Scheme of information flow. Agents travel to users' computers to play games. Those with poorest performances are eliminated. A novelty engine creates new players. The better ones are added to the population, filling the empty slots.

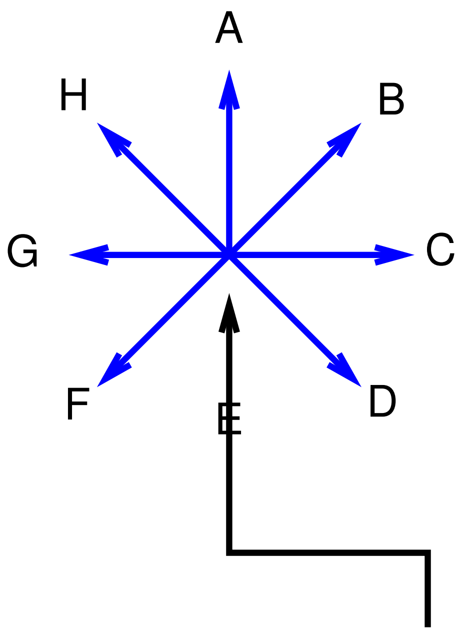


Figure 3.5: A Tron agent perceives the environment through eight distance sensors.

sensors and evaluates its s-expression every third time step: if a RIGHT or LEFT function is output, the agent makes the corresponding turn; otherwise, it will keep going straight.

The simple sensory capabilities imply that an agent's view of the game is quite restricted: the position of the opponent is unknown, and so is the complete view of the game situation.

Tron agents have no state variables, so the behavior of the agent is purely reactive, based solely on 7 distance sensors[1]. Whereas humans may base their game decisions on topological considerations (e.g. "is this region open or closed?"*)*, or follow plans, the complexities of a robot's behavior must emerge from the interaction with the opponent along the game.

3.3.4 Java Applet

When a visitor opens the Tron web page[2] , her browser loads and starts a Java applet. The applet (fig. 3.2) receives the GP code for an agent from our web server and uses it to play one game with the human user. The human moves by pressing the arrow keys and the agent, by evaluating its s-expression. When the game ends, the applet reports the result (win or

[1]Sensor "E", which point backwards, sees the agent's own trace as an immediate obstacle, so it always returns 1.0.

[2]http://www.demo.cs.brandeis.edu/tron

loss) to the server, and receives a new agent for the next game. This cycle continues until the human stops playing.

Our Java Tron application consists of three modules: the *Arena* updates players position and direction, and their traces, updating the display; the *Human Player* module listens to keystrokes and changes the Human's orientation accordingly; a *GP Player* module computes sensor values and feeds them to a GP interpreter that evaluates an agent's s-expression. This module contacts the server over the web, receiving GP code and sending back game results.

3.3.5 Evolving Agents: The Tron Server

The Tron system maintains a population of 100 "live" agents on the foreground server. For each game, an agent is drawn at random from it. The game results are stored in a database. A generation lasts until all 100 agents have played a minimum number of games: new agents play at least 10 games, while veterans from previous generations play only 5 games.

With a proportion of 90 veteran agents and 10 rookies on the population, a generation consists of approximately 450 games by veterans and 100 by novices, thus untested agents play about 18% of all games.

When all agents have completed their minimum number of games, the current generation finishes: agents are sorted by fitness; the worst 10 are eliminated and replaced by 10 fresh ones, supplied by a the *novelty engine*. A new generation begins (fig. 3.4).

Pseudocode of the foreground Tron server

1. `Start with an agent population` A `of 100 robots.`
2. `For each` $a \in A$ `let` $c_a = 0$
3. `(Loop)`
   `While` $\min\{c_a, a \in A\} < 10$, `wait for events:`
   (a) `On event that a Java applet requests a game over the Internet,`
   `Select an agent` $a \in A$ `with probability`
   $$P(a) = \frac{w(a)}{\sum w(x), x \in A}, w(a) = \max\{1, 10 - c_a\}$$
   `and send it to the applet.`
   (b) `On event that an applet reports the results of a game between human` h
   `and agent` a,
   `Save in database:`
   `Game result (win, tie, or lose); human id, agent id, time stamp, and`

```
    list of moves.
    If a ∈ A then let c_a = c_a + 1
```

4. `Sort` A `according to fitness,` $A = a_1, \ldots, a_{100}$, `and let` $V = a_1, \ldots, a_{90}$

5. `Fetch 10 new agents from novelty engine and call them` R

6. `For each` $a \in V$ `let` $c_a = 5$.
   `For each` $a \in R$ `let` $c_a = 0$
   `Let` $A = V \cup R$

7. `Go to (Loop)`

3.3.6 Fitness Function

Defining an appropriate fitness measure to rank our agents has proven difficult. In principle we defined a variant of fitness sharing [8] by giving points for doing better than average against a human player, and negative points for doing worse than average. The fitness of agent *a* was defined as:

$$F(a) = \sum_{\{h:p(h,a)>0\}} \left(\frac{s(h,a)}{p(h,a)} - \frac{s(h)}{p(h)} \right) \left(1 - e^{\frac{p(h)}{10}} \right) \tag{3.1}$$

where $s(h,a)$ is the number of games lost minus the number of games won (*score)* by a human opponent h against a; $p(h,a)$ is the total number of games between the two; $s(h)$ is the total score of h; and $p(h)$ is the number of games that h has played. All games played are counted, not just those that belong to the current generation. The factor $\left(1 - e^{\frac{p(h)}{10}}\right)$ is a confidence measure that devalues the average scores obtained against humans who have played only a small number of games.

A second part of the experiment assayed a new definition of fitness, based on our statistical analysis of players' strengths. This problem is discussed in detail in section 3.6.

3.3.7 Novelty Engine

Evolutionary algorithms create new entities and evaluate their fitness, introducing variations on the existing population through the use of crossover and mutation operators. Such recombinations often yield uninteresting individuals, identical to or worse than their parents. Typically, evaluations are rapid and unfit children are quickly filtered out. However, in our case, this approach would be wasteful since evaluation is obtained from a sparse resource — precious interactions between agents and humans.

The Tron architecture uses a separate *novelty engine* as the source of new individuals. This module coevolves a population of 1000 agents by playing them against each other. The best robots are chosen to be incorporated into the main population. Even though self-play does not provide enough information to know which strategies will perform best against people, this method is much better than blind recombination for creating interesting new agents.

The novelty engine is a continuously running generational GA with 50% elitism. Every agent in the population plays against a training set T of $t = 25$ robots. Fitness is evaluated, and the bottom half of the population is replaced by random mating with crossover of the best half. The fitness function is defined as follows:

$$F_T(a) = \sum_{\{a' \in T : pt(a,a')>0\}} \frac{pt(a,a')}{l(a')} \tag{3.2}$$

where T is the training set, $pt(a, a')$ = {0 if a loses against a', 0.5 if they tie and 1 if a wins} and $l(a')$ is the number of games lost by a'. Thus we give more points for defeating good players than bad players.

The training set consists of two parts. There are f fixed members which come from the foreground process. The remaining $t - f$ members of the training set are replaced each generation with a fitness sharing criteria. The new training set T' is initialized to the empty set and then new members are added one at a time, choosing the highest according to the following shared fitness function:

$$F_{T,T'}(a) = \sum_{a' \in T} \frac{pt(a,a')}{(1 + \sum_{a'' \in T} pt(a'',a)} \tag{3.3}$$

This selection function, adapted from Rosin [118], decreases the relevance of a case that has already been "covered", that is, when there is already a player in the training set that beats it.

At the end of a generation, the bottom half of the population is dropped, and so is the (changing) part of the training set. 500 new agents are created by (uniform) random mating (crossover), with a mutation rate of 0.04 (a parent's subexpression has a 4% chance of being replaced by a random new expression instead).

Feedback from Main Population to Novelty Engine

With the main population/novelty engine setup, the idle time — while the system is patiently waiting for human games to come in[3] — is devoted to the fabrication of the best

[3] Averaging 100 thousand games per year (0.2 per minute), the pace of human-agent interaction is between 3 and 4 orders of magnitude slower than our C code, capable of playing about 1000 games per minute.

opponents we can come up with.

It is expected that an evolutionary algorithm should increase the quality of the new entities with every each generation. If there was no feedback from the front end back to the novelty engine, the latter would remain oblivious to the information being collected by the selective process acting on the foreground. We have proposed and implemented two ways for surviving agents to come back and reproduce in the novelty engine.

1. *Using them as trainers.* A reasonable setting for coevolving Tron agents by self-play alone would have $f = 0$. By setting $f = 15$, some of the champions-against-people come back to the novelty engine to act as trainers. The hope is that, by transference, new agents are favored which are themselves good against people.

2. *Reintroducing them in the population.* Successful agents can be simply reintroduced in the population to let them compete with the other coevolving robots and reproduce, provided they are successful against their peers.

Tunable Parameters

The novelty engine has three parameters that have changed at different points in the experiment.

- MAXGEN. Every time the foreground server ends a generation, it fetches ten new rookies, the current champions-vs.-robots, to become part of the main population. To avoid the effects of convergence, which could lead to the same agents being sent repeatedly, the background population is restarted every once in a while. The novelty engine checks the number of generations g that the present population has been evolving for. If $g >$ MAXGEN, then the present population is killed and evolution starts with a fresh random population.

- *f*. When the novelty engine restarts, it fetches f new agents from the foreground that become the fixed part of the training set.

- SEED. This is a boolean parameter. Upon restart of the coevolutionary run, either 1000 new random agents are created or, if SEED is true, the current 100 champions-against-people are reintroduced along with 900 random ones — the foreground population is used as a seed for the background.

Pseudocode of the Novelty Engine

1. `(Reset)`
   `Create a population` $P = \{p_1, \ldots, p_{1000}\}$ `of random robots`
   `and a random training set` T_2 `of` $t - f$ `agents.`

2. (Seed)
   If option SEED is true,

   (a) fetch 100 best from main population
   (b) replace $\{p_{901}, \ldots, p_{1000}\}$ with them

3. (Refetch)
   Fetch f best agents from the main server
   and call this group T_1

4. Let $g = 0$

5. Repeat forever

   (a) Let $T = T_1 \cup T_2$
   (b) Play each $a \in P$ against each $a' \in T$
   (c) Sort $P = \{p_1, \ldots, p_{1000}\}$ according to eq. (3.2)
   (d) For $i = 1$ to 500
       Select random $a_1, a_2 \in \{p_1, \ldots, p_{500}\}$ and
       replace p_{i+500} with a random crossover of a_1 and a_2
   (e) Let $T' = \varnothing$, then for $i = 1$ to $t - f$
       add a new agent to T' by eq. (3.3).
       Let $T_2 = T'$
   (f) Let $g = g + 1$
   (g) If the main population has finished a new generation, then
       i. Send $\{p_1, \ldots, p_{10}\}$ to main population as next group of rookies
       ii. If $g >$ MAXGEN then go to (Reset) else goto (Refetch)

3.4 Results

Our server has been operational since September 1997; we have collected the results of all games between agents and humans; the system is still running. The results presented in this section are based on the first 525 days of data (204,093 games). A total 4037 human players and 3512 agent players have participated, each of them having faced just some of all potential opponents (fig. 3.6). The “aftermath” section (3.6) discusses a new configuration of the system and the results obtained with it.

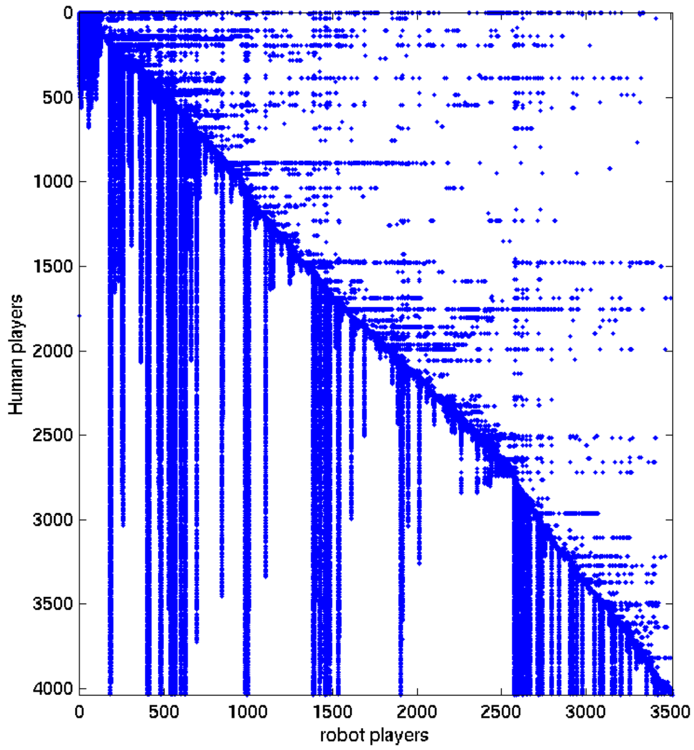


Figure 3.6: Who has played whom: A dot marks every human-robot pair who have played each other at least once. Both populations are sorted by the date of their first appearance. The long vertical lines correspond to robots that have been part of the population for a long time, and thus have played against most newcomers.

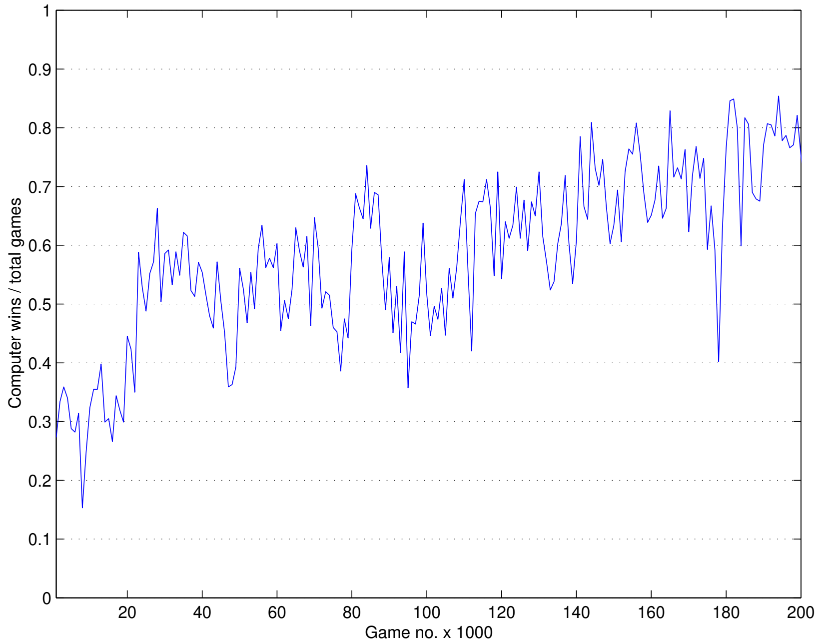


Figure 3.7: Evolution of the win rate: WR sampled in groups of 1000 games.

3.4.1 Win Rate (WR)

An intuitive performance measure is the *win rate* (WR), which is the fraction of games that the artificial players win;

$$\mathrm{WR} = \frac{\text{games won}}{\text{games played}} \tag{3.4}$$

The average win rate over the total number of games played is 0.55, meaning that 55% of all games completed resulted in agent victories.

The WR has been changing over time (fig. 3.7), in an oscillating fashion. This noisy behavior is a natural phenomenon in a coevolutionary environment, and occurs here more noticeably since one of the evolving populations consists of random human players. Each of the 4037 persons sampled here has a different level of expertise and has played a different number of games (another variable factor is the speed of the game on the user's machine, which may have a slower pace when the Java environment is too slow[4]).

There is a visible trend toward improvement. Whereas at the beginning of our experiment, the Tron system won about 30% of its games, by the end of the period it wins about 80% of its games. This is a strong indication of the improvement of the system's perfor-

[4]Our Java Tron uses a millisecond sleep instruction to pace the game, but different implementations of the Java Virtual Engine, on different browsers, seem to interpret it with dissimilar accuracies. The effect is more noticeable on machines with slow CPUs and old browsers.

mance.

An increasing WR could be hiding other phenomena besides improvement in the quality of game of the Tron system. Humans could be playing worse now than before, for example, for whatever the reasons; or novices could be dominating the human population and playing the bulk of games. In the next section we describe a statistical technique that yields a much finer measure, avoiding the pitfalls of WR.

3.4.2 Statistical Relative Strength (RS)

To further analyze performance and learning within the Tron system, we employ a paired-comparisons maximum likelihood model.

Paired comparisons models are statistical methods that estimate the relative strengths or preferences of a group of participants. The "Elo ratings" for Chess, conceived by A. Elo [34] are one example of such method. Chess poses some problems akin to ours, as one would like to ask, say, "was Capablanca better than Fisher?" Even if the two players did play each other, one might not have been at the peak of his abilities at the time. All the information from opponents they played in common, and how well they performed, should be put together. We have followed the maximum likelihood approach described by Joe [73], applied by the author to the Chess problem among others.

Elo's model — used today for many other games, including the so-called "game ladders" — assigns a low ranking to a novice, who can slowly climb up as she wins games against other ranked players. Maximum likelihood statistics such as Joe's are better suited to our problem because they compute the most feasible ranking for all players, without presuming that young ones are bad.

Paired Comparisons Analysis

The goal of paired comparison statistics is to deduce a ranking from an uneven matrix of observed results, from which the contestants can be sorted from best to worst. In the knowledge that crushing all the complexities of the situation into just one number is a large simplification, one wishes to have the best one-dimensional explanation of the data.

Each game between two players (P_i, P_j) can be thought of as a random experiment where there is a probability p_{ij} that P_i will win. Games actually observed are thus instances of a binomial distribution experiment: Any sample of n games between P_i and P_j occurs with a probability of

$$P(\text{sample}) = p_{ij}^{w_{ij}}(1 - p_{ij})^{n-w_{ij}} \tag{3.5}$$

where w_{ij} is the number of wins by player P_i.

We wish to assign a *relative strength* (RS) parameter λ_i to each of the players involved in a tournament, where $\lambda_i > \lambda_j$ implies that player P_i is better than player P_j.

A probability function F such that $F(0) = 0.5$ and $F(x) = 1 - F(-x)$ (for all $x \in \Re$) is chosen arbitrarily; following [73] we use the logistic function

$$F(x) = \frac{1}{1+e^{-x}} \tag{3.6}$$

The model describes the probabilities p_{ij} as a function of the RS parameter λ_i for each player:

$$p_{ij} = F(\lambda_i - \lambda_j) \tag{3.7}$$

so the outcome of a game is a probabilistic function of the difference between both opponent's strengths. The conditions imposed on F imply that players with equal strength are estimated to be equally likely to win or lose, and that the probability of P_i winning is equal to that of P_j losing.

The observed data is a long sequence of games between opponent pairs, each one a either a win or a loss. According to eq. 3.7, the probability of that particular sequence was

$$P = \prod_{i,j} F(\lambda_i - \lambda_j)^{w_{ij}} \left(1 - F(\lambda_i - \lambda_j)\right)^{n_{ij} - w_{ij}} \tag{3.8}$$

for any choice of λ_i's.The set of λ_i's that best explains the observations is thus the one that maximizes this probability. The well known method of maximum likelihood can be applied to find the maximum for eq. 3.8, generating a large set of implicit simultaneous equations on $\lambda_1, \ldots \lambda_M$ that are solved by the Newton-Raphson algorithm.

An important consideration is, the λ_i's are not the true indeterminates, for the equations involve only paired differences, $\lambda_i - \lambda_j$. One point has to be chosen arbitrarily to be the zero of the RS scale.

A similar method permits assigning a rating to the performance of any smaller sample of observations (one player for example): fixing all the λ_i's on equation (3.8), except one, we obtain

$$\text{wins} = \sum_i F(\lambda - \lambda_i) \tag{3.9}$$

where λ is the only unknown — all the other values are known. The single indeterminate can be found with identical procedure.

A player's history of games is a vector $(x_1, \ldots x_N)$ of win/loss results, obtained against opponents with known RS's $\lambda_{i_1}, \ldots, \lambda_{i_N}$, respectively. Eq. (3.9) can be solved iteratively, using a "sliding window" of size $n < N$, to obtain strength estimates for $(x_1, \ldots x_n)$, then for $(x_2, \ldots x_{n+1})$, and so on. Each successive value of λ estimates the strength with respect to the games contained in the window only.

With this window method we can do two important things: analyze the changing performance of a single player over time, and, putting the games of a group of players together into a single indeterminate, observe their combined ranking as it changes over time.

Altogether, the paired comparisons model yields:

- A performance scale that we have called Relative Strength (**RS**). The zero of the scale is set arbitrarily (to the one of a fixed sample player: agent 460003).
- An ordering of the entire set of players in terms of proficiency at the game, as given by the RS's.
- An estimation, for each possible game between two arbitrary players, of the win-lose probability (eq. 3.7). With it, an estimation of exactly how much better or worse one is, as compared to the other.
- A way to measure performance of individuals or groups over time.
- A possible fitness measure: the better ranked players can be chosen to survive.

Paired comparisons statistics are an open research area. Desirable improvements include: better accuracy, lower computational costs, estimation of error, and time sensitivity. New models such as Glickman's [54] may offer improvements on those areas.

3.4.3 Analysis of Results

In an effort to track the Red Queen (see section 3.6), without having to play games outside those involved in the coevolutionary situation, we can think of each player as a relative reference. In Tron, each agent has a fixed strategy and thus constitutes a marker that gives a small amount of evaluation information. A single human, as defined by their login name and password, should also be relatively stable, in the short term at least. The paired comparisons model described above is a powerful tool that uses the information of all the interwoven relationships of a matrix of games (fig. 3.6) at the same time. Every player, with his/her/its wins and loses, contributes useful bits of information to evaluate all the rest.

There are degenerate situations where the present model would give no answer. If one has knowledge of games between players A and B for example, and also between C and D, but nor A nor B have ever played C or D, there is no connectivity, and consequently no solution to equation 3.8. In the Tron case, connectivity is maintained throughout the experiment by the multitude of players who come and go, coexisting for a while with other players who are also staying for a limited time. The whole matrix is connected, and the global solution propagates those relative references throughout the data set.

An increasing WR (section 3.4.1) is not a whole proof that our system was evolving towards better agents. It could be the case, for example, that humans became increasingly sloppy, losing more and more of their games while agents stayed more or less the same.

(a) Best Players

	Strength	Player ID
1.	3.55	887
2.	1.60	1964
3.	1.26	388
4.	1.14	155
5.	1.07	1636
6.	1.05	2961
7.	0.89	3010008
8.	0.89	3100001
9.	0.84	1754
10.	0.81	2770006
11.	0.81	3130004
12.	0.76	2980001
13.	0.70	1860
14.	0.66	2910002
15.	0.62	3130003

(b) Worst Players

	Strength	Player ID
1.	-4.64	2407
2.	-3.98	2068
3.	-3.95	3982
4.	-3.88	32
5.	-3.75	1986
6.	-3.73	33
7.	-3.69	3491
8.	-3.41	2146
9.	-3.39	2711
10.	-3.36	3140
11.	-3.31	1702
12.	-3.31	1922
13.	-3.30	2865
14.	-3.27	2697
15.	-3.22	2441

Table 3.1: Best players and worst players lists. Only players with 100 games or more are been considered (244 humans, 391 robots). ID numbers greater than 10000 correspond to artificial agents.

Applying the paired comparisons model gave us more reliable information. We computed the RS for every computer and human player[5]. For the first time we were able to compare agents and humans with each other: fig. 3.1 lists the 15 best (a) and worst (b) players. Each human and robot is labelled with a unique ID number: humans were numbered consecutively by their first appearance, and robots have id numbers all greater than 10,000 (the first 3 digits encode the generation number).

The top players table (fig. 3.1a) has 6 humans at the top, the best agent so far being seventh. The best player is a human, far better than all others: according to eq. 3.7, an estimated 87% chance of beating the second best!. This person must be a genius.[6]

The difference between the top group of human players (RS around 1.1) and the top agent players (RS's around 0.7) is about 60%. Seven out of the best 15 players are agents. The best agent, R. 301008, is estimated to be better than 97.5% of the human population.

[5]Players who have either lost or won all their games cannot be rated, for they would have to be considered infinitely good or bad. Such players convey no information whatsoever to rank the others. Losing against a perfect player, for example, is trivial and has no information contents. Perfect winners/losers have occurred only on players with very little experience. There is one human (no. 228) who won all 37 games he/she played. Should we consider him/her the all-time champion? Perhaps. The present model does not comprehend the possibility of a perfect player. To eliminate noise, we only consider players with 100 games or more. All "unrated" players are below this threshold.

[6]Or perhaps, a user with an old computer, running the applet below its normal speed.

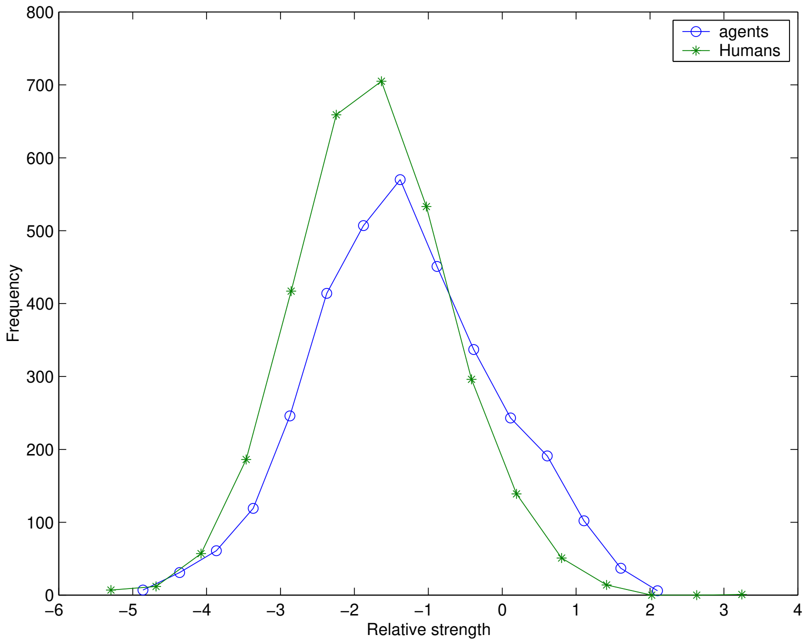


Figure 3.8: Strength distribution curves for agents and humans.

The worst players table (fig. 3.1b) is composed of all humans. This does not indicate that all agents are good but rather, that most bad agents are eliminated before reaching 100 games.

3.4.4 Distribution of Players

The global comparative performance of all players can be seen on the distribution curves (fig. 3.8). Here we have plotted all rated players, including those with just a few games. The fact that some agents curve have large RS values indicates that the coevolutionary engine that produces new Tron players, has managed to produce some players that are good against people. But at the same time, the wide spread of agent levels, from very bad to very good, shows us that there is a reality gap between playing against other robots and playing against humans: all agents that ever played against humans on the website were selected among the best from an agent-agent coevolutionary experiment that has been running for a large number of generations: our novelty engine. If being good against agents was to guarantee that one is also good against people, robots would not cover a wide range of capacities: they would all be nearly as good as possible, and so would fall within a narrow range of abilities.

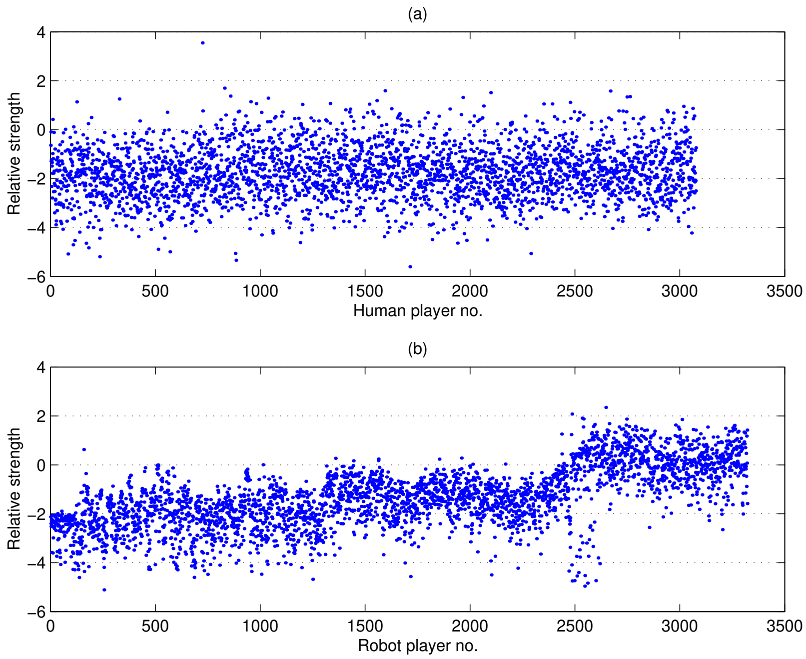


Figure 3.9: New humans (a) are about as good as earlier ones on average. New robots (b) may be born better, on average, as time passes, benefiting from feedback from agent-human games and improvements on the configuration of the novelty engine.

3.4.5 Are New Generations Better?

It seems reasonable to expect that new humans joining the system should be no better, nor worse, on average, than those who came earlier. This is indeed the case, according to the data on fig. 3.9a: both good and not-so good people keep joining the system. Tron agents (fig. 3.9b) do show differences.

Our attempt for progressively increasing the quality of new agents produced by the novelty engine, by having them train against those best against humans, was partially successful: graph 3.9b shows a marginal improvement on the average strength of new players, first to 2500-th. But noticeable better agents beginning at 2800 come to confirm the previous findings of other researchers [4, 130] in the sense that the coevolving population used as fitness yields more robust results than playing against fixed trainers who can be fooled by tricks that have no general application. This point is discussed in detail in section 3.7.

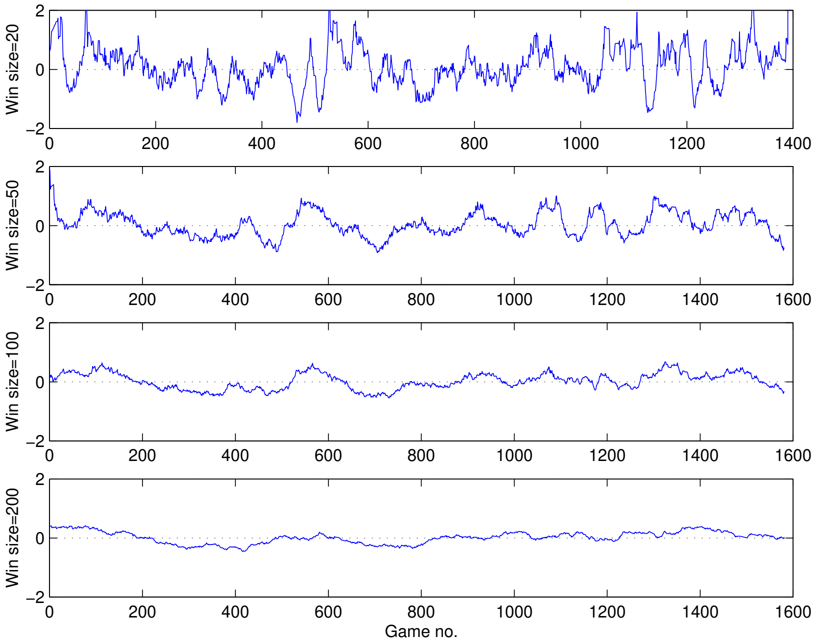


Figure 3.10: Performance of robot 460003 — which was arbitrarily chosen as the zero of the strength scale — observed along its nearly 1600 games, using increasingly bigger window sizes.

3.5 Learning

We wish to study how the performance of the different players and species on this experiment has changed over time. Fig. 3.10 shows the sliding window method applied to one robot. It reveals how inexact or "noisy" the RS estimates are when too few games are put together. It is apparent that 100 games or more are needed to obtain an accurate measure.

Since each individual agent embodies a single, unchanging strategy for the game of Tron, the model should estimate approximately the same strength value for the same agent at different points in history. This is indeed the case, as seen for example on figs. 3.10 (bottom) and 3.11a.The situation with humans is very different: people change their game, improving in most cases (fig. 3.11b).

3.5.1 Evolution as Learning

The Tron system was intended to function as one intelligent, learning opponent to challenge humanity. The strategy of this virtual agent is generated by the random mixture of Tron robots in the evolving population; 18% of the games being played by new, untested agents, exploring new strategy space. The remaining games are played by those agents considered the best so far — survivors from previous generations, exploiting previous knowledge. In

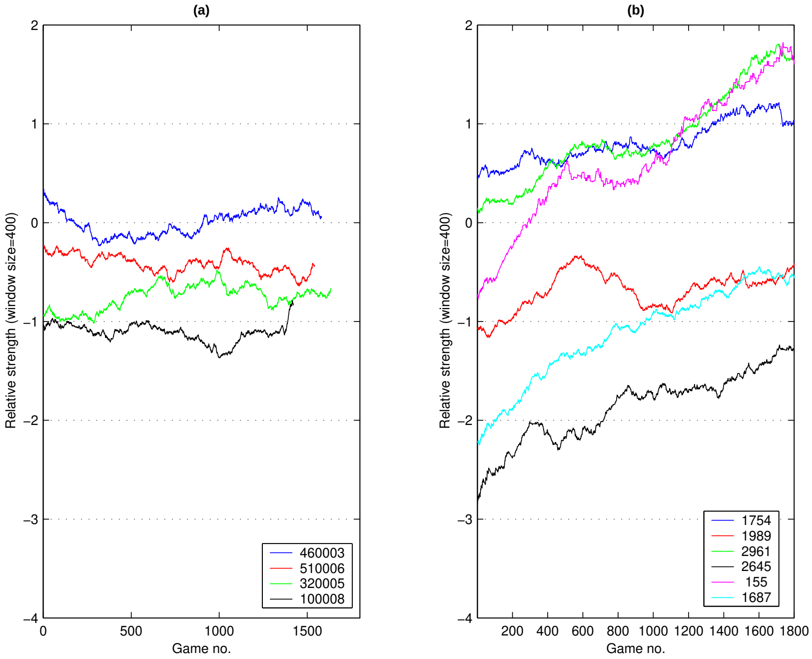


Figure 3.11: (a) Robot's strengths, as expected, don't change much over time. Humans, on the other hand, are variable: usually they improve (b).

terms of traditional AI, the idea is to utilize the dynamics of evolution by selection of the fittest as a way to create a mixture of experts that create one increasingly robust Tron player.

Solving equation (3.9) for all of the computer's games put together yields the performance history of the whole system considered as a single playing entity. Fig. 3.12 shows that the system has been learning throughout the experiment, at the beginning performing at a RS rate below –2.0, and at the end around 0.

But the RS is an arbitrary scale: what does it mean in human terms? The next graph re-scales the RS values in terms of the percent of humans below each value. *Beginning as a player in the lower 30 percent, as compared to humans, the Tron system has improved dramatically: by the end of the period it is a top 5% player (fig. 3.13).*

3.5.2 Human Behavior

Why would humans want to continue playing Tron? One mechanism we devised for attracting "web surfers" and enticing them to keep coming back is our "Hall of Fame" (fig. 3.14). The hall of fame, or ranking, is a form of bringing up competition between humans into a game that otherwise is played in isolation.

Figure 3.15 shows that we have consistently attracted groups of veteran players along

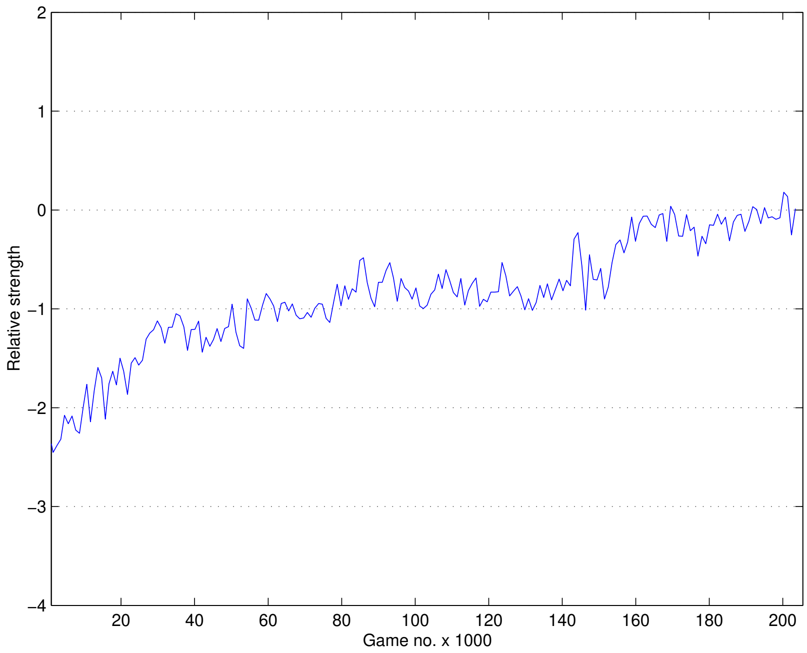


Figure 3.12: Relative strength of the Tron species increases over time, showing artificial learning.

with a steady stream of new participants. The presence of seasoned humans helps us get an accurate evaluation of agents, but novices were necessary, at least in the beginning, to discriminate between agents who could have lost all games against an expert. Figure 3.6 shows horizontal lines which represent some of the veterans, coming back again and again over long time spans. The more such players are present, the more accurate is the computation of RS indexes.

Is the human species getting better as well? No. Redoing the same exercise of figure 3.12, but now tracing the strength level of all human players considered as one entity, we obtain a wavy line that does not seem to be going up nor down (fig. 3.16). This shows that, although individual humans improve, new novices keep arising, and the overall performance of the species has not changed over the period that Tron has been on-line.

An altogether different image emerges when we consider humans on an individual basis. Although a large number of games are needed to observe significant learning, there is an important group of users who have played 400 games or more. On average, these humans raise from a performance of –2.4 on their first game, to –0.8 on their 400th game, improving approximately 1.5 points over 400 games (fig. 3.17). The learning rate is dramatically faster for humans, compared to the approximately 100,000 games (against people) that our system needed to achieve the same feat (fig. 3.12).

On fig. 3.18 we have plotted the learning curves of the 12 most frequent players. Many

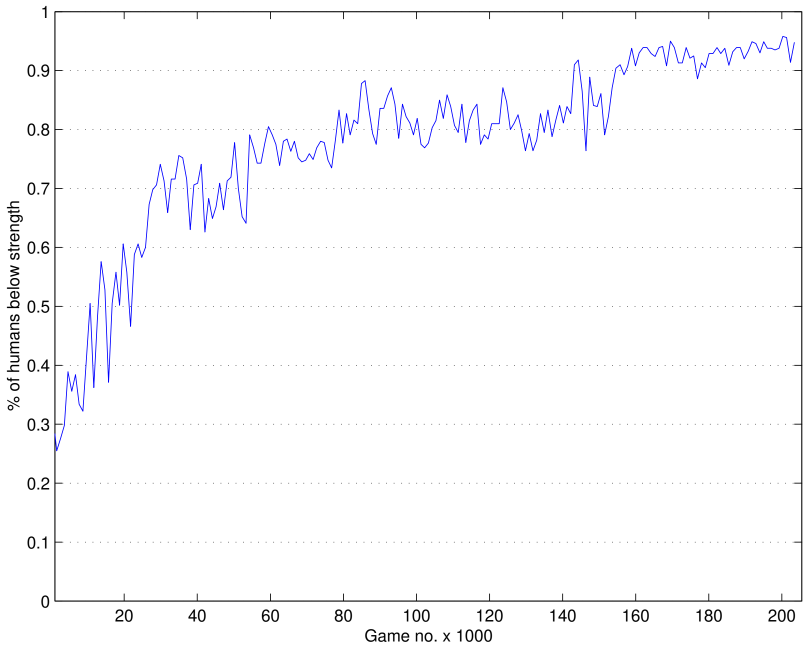
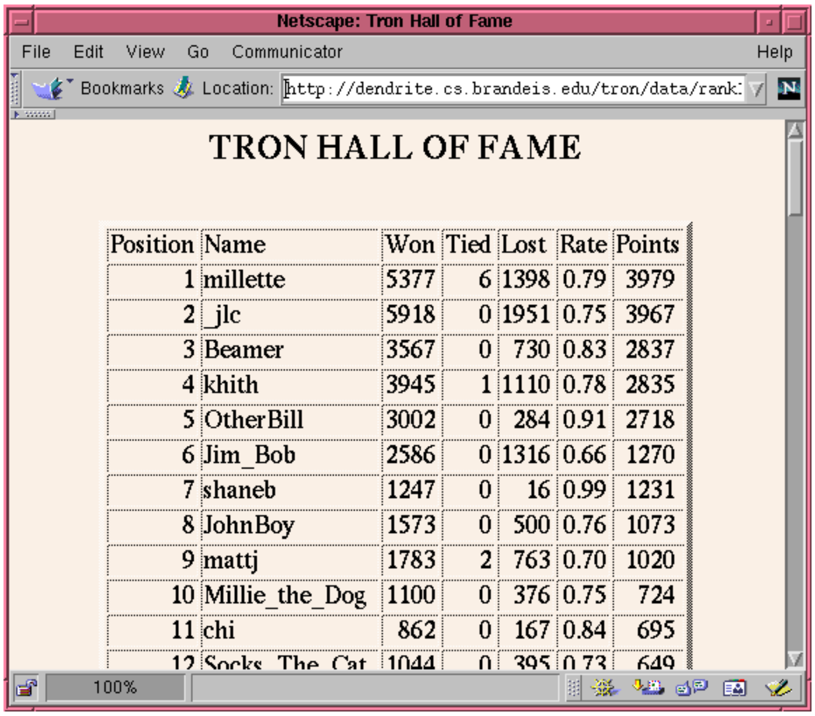


Figure 3.13: Strength values for the Tron system, plotted as percent of humans below. In the beginning our system performed worse than 70% of all human players. Now it is within the best 5%.



TRON HALL OF FAME

Position	Name	Won	Tied	Lost	Rate	Points
1	millette	5377	6	1398	0.79	3979
2	_jlc	5918	0	1951	0.75	3967
3	Beamer	3567	0	730	0.83	2837
4	khith	3945	1	1110	0.78	2835
5	OtherBill	3002	0	284	0.91	2718
6	Jim_Bob	2586	0	1316	0.66	1270
7	shaneb	1247	0	16	0.99	1231
8	JohnBoy	1573	0	500	0.76	1073
9	mattj	1783	2	763	0.70	1020
10	Millie_the_Dog	1100	0	376	0.75	724
11	chi	862	0	167	0.84	695
12	Socks_The_Cat	1044	0	395	0.73	649

Figure 3.14: Snapshot of the Tron "Hall of Fame" web page (9/4/00). Players win a point for every game won, and lose one per game lost. We encouraged achieving immortality by playing more games, and winning, rather than having the best winning rate.

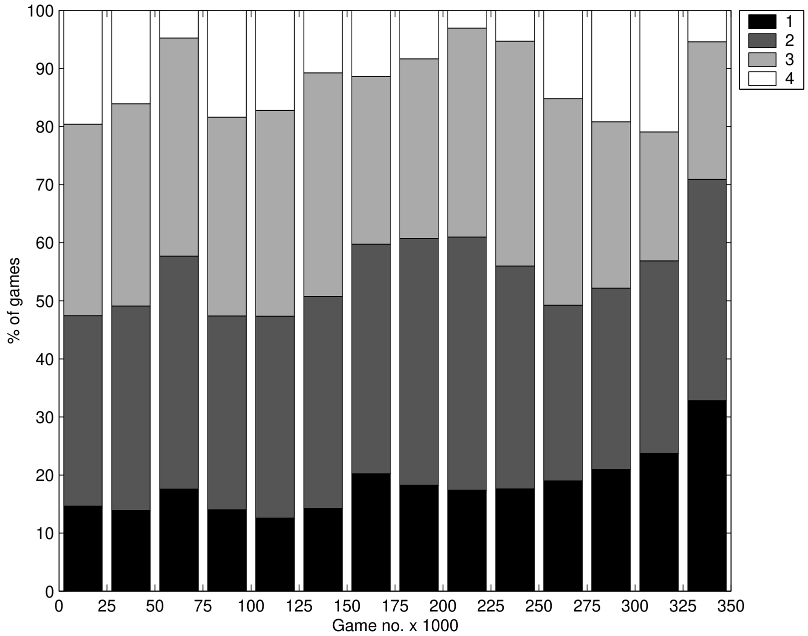


Figure 3.15: Composition of human participants. Human players were divided in four groups: novices, who have played up to 10 games (1); beginners, 11 to 100 (2); seasoned, 101 to 1000 (3); and veterans, more than 1000 (4). From the beginning of the experiment all four groups represent large parts of the totality of games.

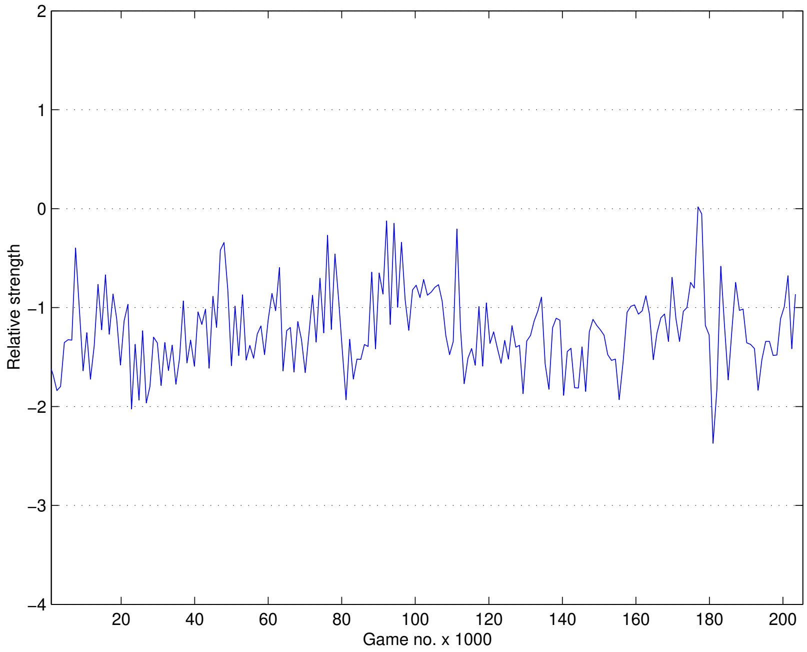


Figure 3.16: Performance of the human species, considered as one player, varies strongly, complicating things for a learning opponent, but does not present overall trends (compare to fig. 3.12).

of them keep learning after 1000 games and more, but some plateau or become worse after some time.

3.6 Measuring Progress in Coevolution

From the Red Queen Effect to Statistical Fitness

Evolution, as trial-and-error based learning methods, usually relies on the repeatability of an experience: Different behavioral alternatives are tested and compared with each other. But agents acting on real environments may not be able to choose which experience to live. Instead, the environment provides varying initial conditions for each trial.

In competitive games for example, it is difficult to compare players with each other if they are not able to choose their opponents. The analysis methodology we adopted for the Tron experiment (section 3.4.2) is a statistics-based approach to solving this problem.

In a coevolutionary environment, the Red Queen Effect [24] makes it difficult to evaluate progress, since the criterion for evaluation of one species is the other, and vice versa. A higher number of wins does not necessarily imply better performance.

Similar problems arise when one tries to compare the performances of past and present

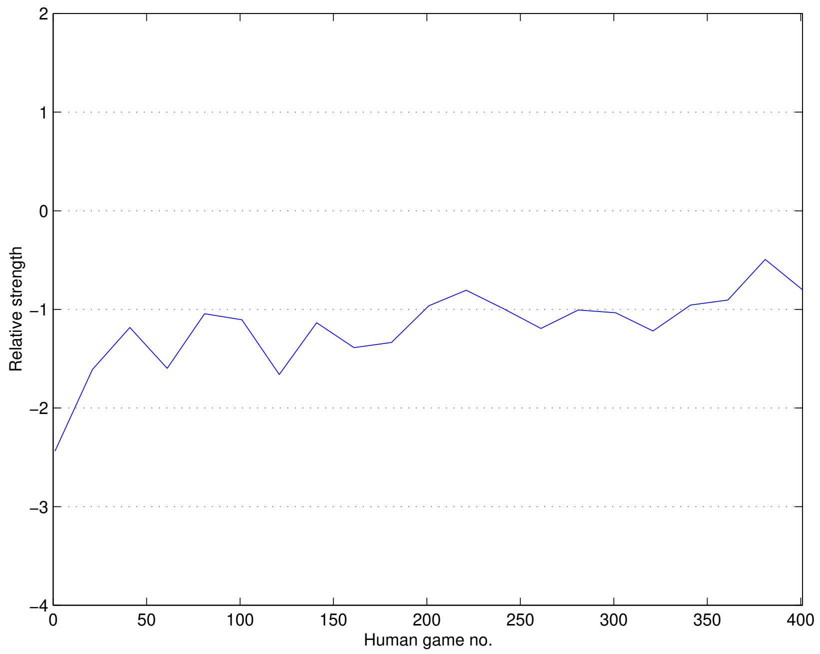


Figure 3.17: Average human learning: RS of players' *n*-th games up to 400. A first-timer has an estimated RS strength of -2.4; after a practice of 400 games he is expected to play at a -0.8 level. Only users with a history of 400 games or more were considered (*N*=78).

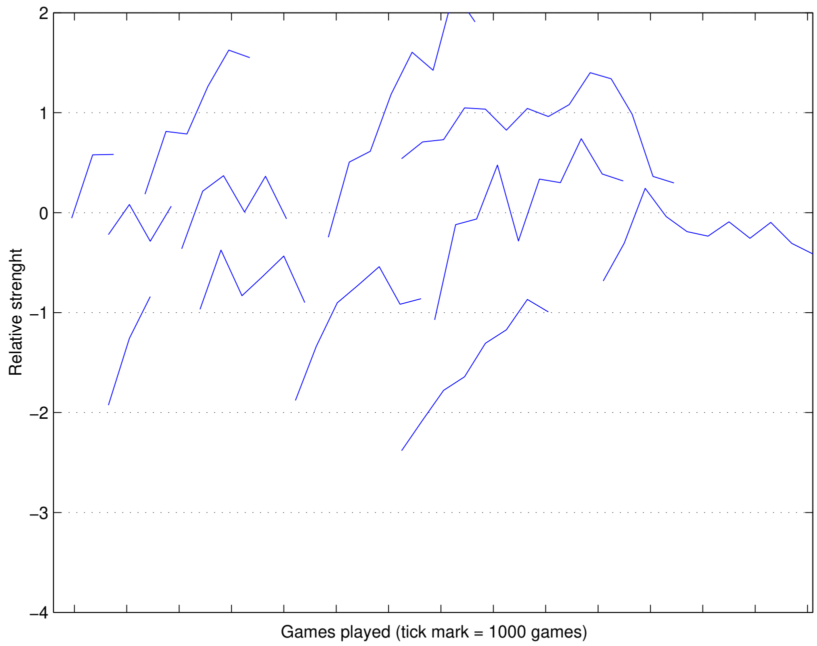


Figure 3.18: Individual learning: strength curves for the 12 most frequent human players (curves start at different x values to avoid overlapping). All users change; nearly all improve in the beginning, but later some of them plateau or descend whereas others continue learning.

players. A well-known strategy for evaluating coevolutionary progress in presence of the Red Queen effect is to take a sample set, an advanced generation for example, and use them to evaluate all players [24, 109]. This is impossible here: we cannot recreate the behavior of humans who played in the past.

Some fixed agents could conceivably be kept in the population for evaluation purposes, but even if one or a few agents were to be present in all generations, most people would play against them only a few times, yielding a measure of low confidence. At the onset of the experiment, we were not willing to sacrifice performance, nor slow down the evolutionary pace by keeping fixed losers inside the population (if they were winners, they would not have to be kept alive artificially, but without an oracle we could not choose them in advance).

Evaluating fitness in coevolution is a closely related problem. The most basic way to assign fitness to players in a competitive/coevolutionary environment is to sum up all wins [4, 6, 65]. More advanced is the use of fitness sharing strategies [8, 75, 118]. Different researchers have tried to reduce the number of games to be played in each generation: large savings can be obtained by matching players against a sample instead of the whole population, "finding opponents worth beating" [119, 123]. The assumption, however, that one can choose the opponents, could not be upheld in our case, where human opponents come and go at will, and an entirely different approach to scoring was needed as well.

The Tron experiment assayed a fitness sharing-inspired fitness function (eq. 3.1 on page 73). We knew that different agents would play against some of the same and some different humans, so simply adding up all wins would not suffice. Instead we compared winning ratios: according to this equation, agents get positive points when they do better than average against a human, and negative points for doing worse than average. The more experienced the human is, the more valuable those points are.

This function was relatively successful in finding good Tron agents, but had problems that we did not foresee. Over time, a strong group of agents formed that were reliably better than average, thus surviving for many generations. As these agents had seen hundreds of humans over their history, and were better than average, even though not necessarily the best, they had too many points to be challenged by newer ones.

The need for a more accurate evaluation of performance in coevolution was thus twofold: not only did we wish to study the evolution of the experiments, comparing today's and yesterday's humans and robots; we were also looking for a better measure to further evolve the artificial population in the future. After two years of Tron we had new insights that prompted us to reconfigure the system:

- Thanks to the "control experiment" (section 3.7) we had what was believed to be a stronger configuration for the novelty engine, with parameters MAXGEN=500 and $f = 1$.

- We began to suspect that our fitness function was failing to select the strongest agents.

There seemed to be a group of robots surviving for many generations, contradicting the intuition that better agents should appear regularly.

- We had a stronger method for evaluating agents, given by the statistical RS measure.

The original definition of fitness (eq. 3.1) failed because players who had been around for a long time had encountered more different humans than other robots. If better than average, even slightly, such an agent would collect many fitness points from a large number of sources. A more inexperienced agent, even winning all of its games, might not gain enough points to be ranked above such a veteran. This is an error in our measurement strategy: how could we discard an agent who has not lost even a single game?

Some of these long-term survivors with high fitness are visible on fig. 3.6 as long vertical lines. Fig. 3.19 shows the relationship between fitness and RS strength. Agents' RS are plotted in the y axis, their fitness on the x axis. Graph (b) shows that there is a main cloud of points with an approximate linear correlation between fitness and strength; there is however (graph a) an important group of agents which deviate from the "main sequence", with higher fitness than they should. Graph (c) on this figure has a column showing the RS values of the top 100 agents with respect to fitness and the top 100 agents strength-wise. We conclude that with the paired comparisons method we have found a better algorithm for choosing our top players.

3.6.1 New Fitness Measure for the Main Population

The logical next step was to implement a new fitness function based on our improved performance measurement. We decided to compute the RS strength of all robots (not just those currently "active" on the population) at the end of each generation. Agents who had won all their games would be assigned an RS of $+\infty$, so they would always be selected.

Beginning at generation 397 (which corresponds to game no. 233,877) the main Tron server computes RS for all players and chooses the top 90 to be passed on to the next generation. So step 4 of the main Tron loop (page 73) is replaced with

4′ `Let` $A = \{a_1, \ldots, a_N\}$ `be the set all agents,`
`past and present, sorted by RS.`
`Let` $V = a_1, \ldots, a_{90}$

thus the paired comparisons model became our fitness function.

The present section analyzes results of nearly a year of the new configuration, spanning up to game no. 366,018. These results include the new configuration of the novelty engine, which produced better new rookies (starting at game no. 144,747) , and the upgraded fitness function — based on paired comparison statistics (starting at game no. 233,877).

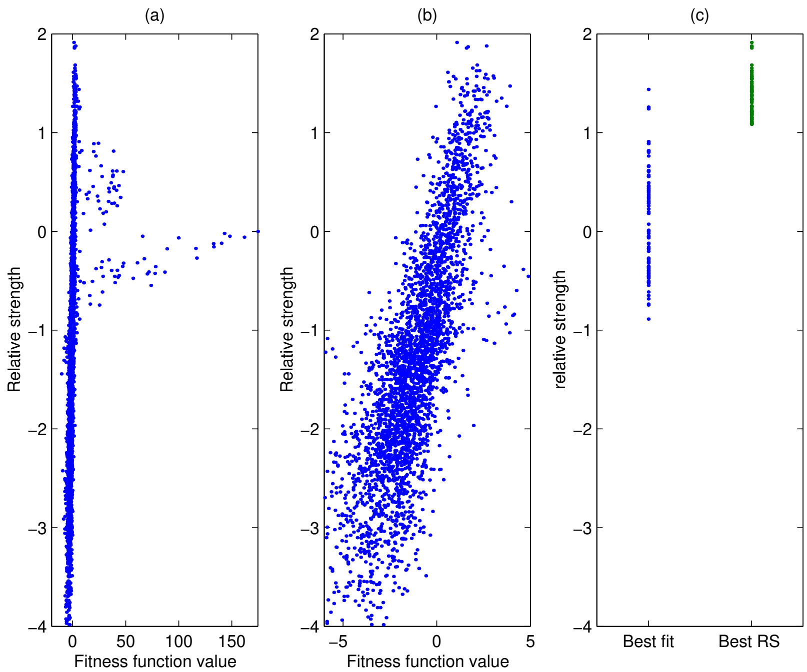


Figure 3.19: Original fitness function vs. statistical strength. (a) fitness value vs. RS for all agents. A group of agents has reached extremely high fitness values even though they are not so special in terms of performance. (b) zooming in on the graph the “main sequence” is apparent. RS and fitness are correlated. (c) the top 100 players according to the fitness formula are different from the top 100 according to the RS.

Fig. 3.20 briefly shows that the system continued learning; both the winning ratio (WR) and the relative strength (RS) went up, whereas the combined human performance stayed about the same.

Fig. 3.21 shows that new humans kept coming with varying strengths, whereas new agents are better since the change of regime on the novelty engine. But there is also a curious flat "ceiling" to the agent's strength graph. In fact this is produced by the selection mechanism: Any agent evaluated above that cutoff will be selected to be in the main population, and kept playing until reevaluation puts it below the top 90.

The main result of this new setup is showing that we have restored a good selection mechanism. This is visible in fig. 3.22: In this graph we have plotted the performance of the system-as-a-whole along with the average strength of new robots being produced at the same time.

The difference between both curves demonstrates the effects of the survival of the fittest brought up by the main Tron server: **the system as a whole performs better than the average agent.**

There is an important increase on the quality of those rookies after game 170,000, with the elimination of the fixed training set and the raise in the number of generations of the novelty engine. At this point, the deficiencies of the original fitness function are evident; between game no. 170,000 and 220,000 there is no performance increase due to selection.

Finally, beginning at game 220,000, selection based on relative strength pushed up once more the performance of the system.

It is too soon to tell whether or not the performance of the Tron system will continue to improve beyond the current state. We feel that we might have reached the limits of agent quality imposed by the representation used.

3.7 Evolving Agents Without Human Intervention: A Control Experiment

We have shown a system that finds good Tron players by coevolving them against humanity. But, were the humans really necessary? In a control experiment, with similar setup but *without human intervention,* we wish to show that the results are not the same — without selection against humans, we would have failed to produce agents who are so good against humans.

3.7.1 Experimental Setup

We ran coevolution between Tron agents and measured the results by using a set of 90 agents, the survivors from generation no. 240. Different settings for the evolutionary pa-

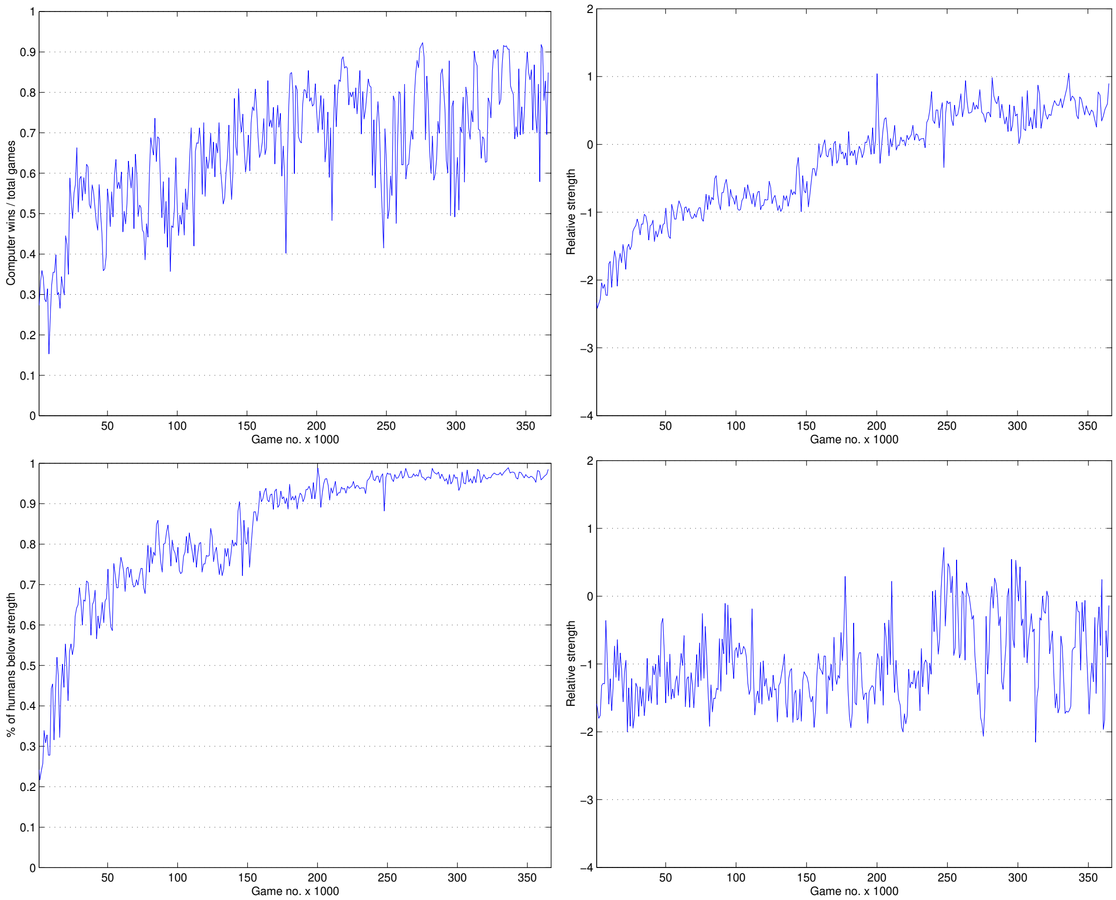


Figure 3.20: Results obtained with the new fitness configuration (beginning at game 234,000) show that the system continued learning. From left to right, top to bottom: Win Rate of the system, RS of the system, RS of the system as compared with humans below, and RS of human population. Compare with figs. 3.7, 3.12, 3.13 and 3.16, respectively.

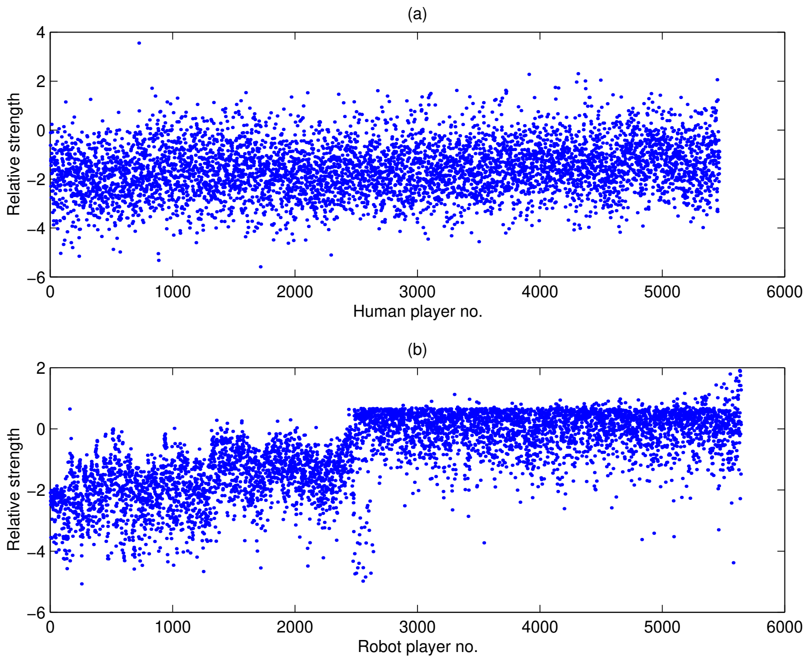


Figure 3.21: Performance of new humans and new agents along time (compare to fig. 3.9) The new configuration of the novelty engine starts producing better robots beginning at robot no. 2500. The flat ceiling of agent's strengths is produced because we are using the same tool for fitness and measurement.

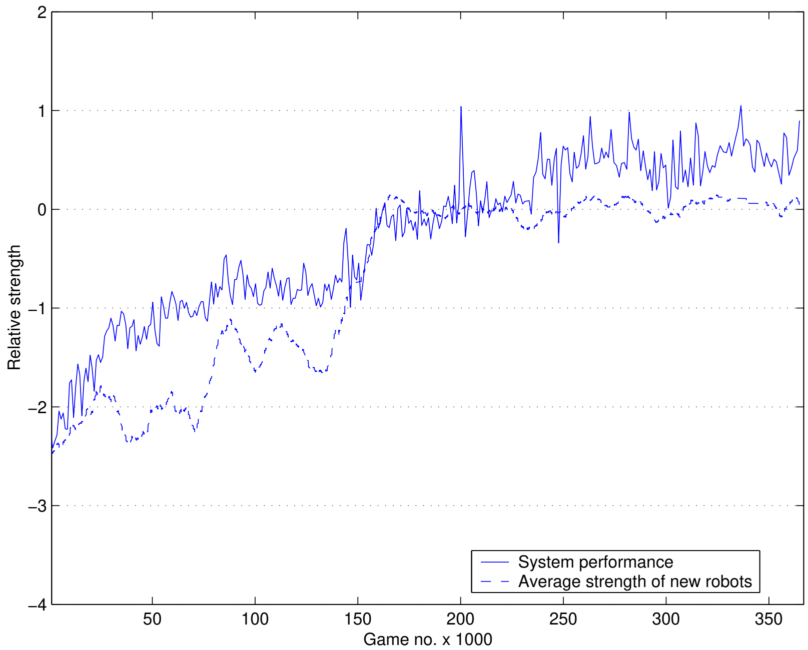


Figure 3.22: Performance of novice robots vs. performance of system as a whole. The robot production engine (broken line) has three stages corresponding to three evolutionary setups. During the first 40,000 games, novices are being produced in short evolutionary runs (20 generations on average). Between 40,000 and 148,000, evolutionary runs are longer (100 generations) and tournaments use 15 fixed champions and 10 evolving champions. From 148,000 and onwards, 24 evolving champions are used for coevolutionary fitness. The increased performance of the system as a whole (solid line) is higher than the average new robots as a results of the selection process. The system stagnated between 148,000 and 220,000 games, when the older fitness function failed to select the better agents. The new statistical fitness function restores the proper behavior (games 220,000-366,000).

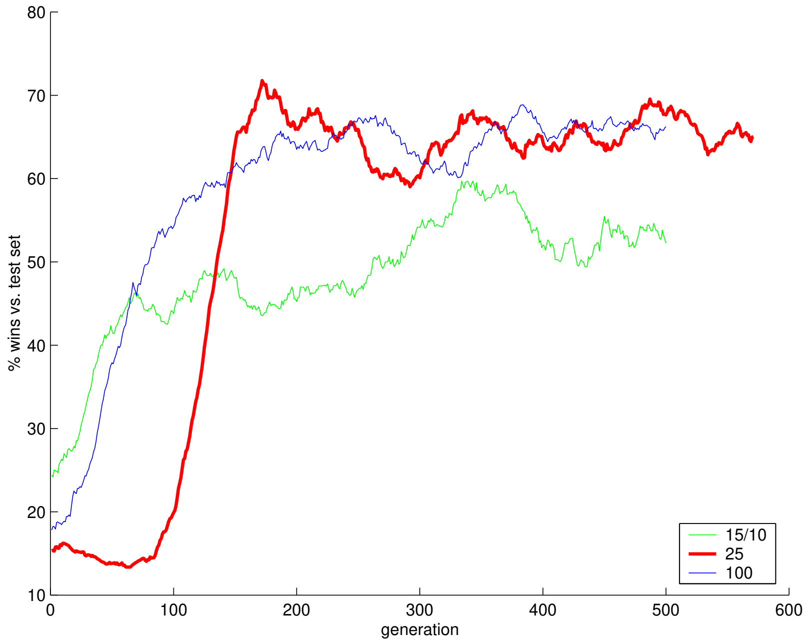


Figure 3.23: *Control Experiments.* Agents were evolved for 500 generations, using self play alone. The performance measure is a set of 90 champions-vs-people (survivors at generation 240). Three different configurations are shown: A with training set size $t = 25$ and full replacement ($f = 0$ (thick line); B with $t = 100$ and $f = 0$ (black line) and C with $f = 15$ and $t = 25$, so 15 out of 25 members of the training set are fixed (grey line). The latter is the configuration originally used by the novelty engine, which was shown here to be suboptimal with respect to the others. This result suggested changing the setup of the novelty engine to the current setting of $f = 1$.

rameters (section 3.3.7) were tested:

- Different training set sizes
- Presence or not of a fixed set of champions

3.7.2 Results

The graph on fig. 3.23 summarizes our findings. Three groups were evolved for 500 generations each: group A with a training set of 25 agents replaced on every generation (as per formula 3.3 on page 74); group B with a larger training set size of 100; and group C with identical settings as the novelty engine: a training set of 25 out of which just 10 are replaced on every generation whereas the remaining 15 remain fixed throughout the experiment, being the 15 best against humans as per equation 3.1.

The results showed group C, with similar setup as the novelty engine, approaching an average 50% score after one or two hundred generations. This is the expected result, since the setup reproduces exactly the conditions over which the evaluation group was produced.

But groups A and B unexpectedly performed *much better*. Both peak at at performance of 65% percent, which means that they are consistently better than the evaluation set. This finding supports previous results by Angeline and Pollack [4] with Tic-Tac-Toe: *evolving against fixed experts is outperformed by coevolution*. In the Tron domain, evolving specialists is a bad idea: they will learn subtle maneuvers that work specifically with the prefixed opponents; but the environment of a changing training set, a moving target, is much broader. In other words, the changing, adaptive nature of a training set replaced with every generation, produces a diverse landscape that results in the evolution of robust solutions.

It was also surprising to observe that there was not much difference with the increased size of the training set on group B. This second group climbs the initial learning ladder in less generations — hitting 50% after 80 generations, as compared with 140 generations for group A. The latter is more efficient, however, given the fact that the number of games played per generation is quadrupled for group B. Both groups settle after reaching a 65% performance, and there was no apparent benefit from choosing a larger training set size.

3.7.3 Tuning up the Novelty Engine

These results prompted a change in parameters in our novelty engine; we decided to reduce f, the size of the fixed part of the training set. Now the bulk of the fitness comes from the coevolving population itself. We keep the one all-time best as the single fixed training example.

The novelty engine has changed configurations twice:

- Initially (robots 1-739), f=15, MAXGEN=0,SEED=false.
  The population is reset every time the front end finishes a generation, and the best 15 against humans are used as trainers.

- Between robots 740 and 2539, f=15, MAXGEN=100, SEED=false.
  The pace of the front end was faster than expected, so the background population was being reset too often; by setting MAXGEN=100 we forced them to continue coevolving for at least 100 generations.

- After robot 2540, f=1, MAXGEN=500, SEED=true.
  With the results of our control, we realized that it takes up to 500 generations for a population to reach a performance plateau; f was reduced to 1 because pure coevolution was shown to outperform evolution against fixed trainers. Instead, the SEED

Control generation no.	Performance vs evaluation set (% wins)	Statistical strength (RS)	Percent of robots below
360	10.0	-4.7	0.1
387	46.7	0.4	80.6
401	54.4	-0.2	59.7
354	61.1	0.4	80.0
541	63.3	0.1	70.5
462	66.7	0.1	67.8
570	70.0	-0.2	60.0
416	75.6	-1.0	40.7
410	78.9	0.4	80.3
535	96.7	0.3	77.2

Table 3.2: Evaluation of control agents (evolved without human intervention) after being introduced into the main population, and evaluated against humans. A robot's performance against our evaluation set does not predict how it will measure up against humans. As a coevolving population wanders through behavior space, it finds both good and bad players.

option was incorporated to provide a means of feedback from the experienced foreground into the novelty engine.

The graph of rookie robots' strengths (fig. 3.9b) shows how the first change introduced a slight improvement, and the second an important improvement in new robots' qualities.

3.7.4 Test Against Humans

To verify the hypothesis that selecting against humanity is not irrelevant, we selected a group of 10 players produced by the control experiment, and introduced them manually in the main population, to have them tested against humans. We ran this generation (no. 250) for longer than our usual generations, to get an accurate measurement.

Table 3.2 summarizes the result of this test. A group of 10 robots was chosen, each one the best from one of the 600 generations that group B (t=100) ran for. We chose the one that performed worst against the evaluation set (generation 360) and the one that performed best (gen. 535), along with eight others, chosen by their different performances vs. the evaluation set.

The last column of the table shows how these robots compare, *as measured by their performance against humans* (RS) with all other ranked robots.

From the internal point of view of robot-robot coevolution alone, all these agents should be equal: all of them are number one within their own generation. If anything, those of later generations should be better. But this is not the case, as performance against a training set suggests that after 100 generations the population is wandering, without reaching higher absolute performance levels. This wandering is also occurring with respect to the human performance space.

We conclude that a coevolving population of agents explores a subspace of strategies that is not identical to the subspace of human strategies and consequently the coevolutionary fitness is different from the fitness vs. people. Without further testing vs. humans, self play alone provides a weaker evolutionary measure.

3.7.5 The Huge Round-Robin Agent Tournament

After 3 years of Tron evolution, there are 5750 ranked robots (some robots are unrated because they lost all their games). Taking advantage of a 16-processor MIMD parallel computer, a round-robin tournament was performed between all robots: 33 million games! How well can this approximate the evaluation against humans?

Figure 3.24 shows the correlation between evaluation vs. humans and evaluation vs. robots. Each dot represents one robot, its winning ratio amongst the 5750 robots on the x axis and its RS against humans on the y axis. To avoid noise on the RS value, only robots who have played 100 games or more were chosen (N=514).

A linear regression (dotted line) tells us what the straight line that best approximates the data is, and the correlation coefficient $R^2 = 0.87$. This means that there is a strong correlation between both values. Comparing with the table that resulted from tests against just 90 robots (table 3.2), the correlation here has improved dramatically.

Even within the limits of our simple sensors and GP operators, the configuration of Tron had the capacity of producing a large diversity of players such that, evaluating against a large number of diverse but highly evolved agents we could predict with a confidence of 90%, how well they will perform against humans.

When the experiment started this was unknown. One might wonder from this result, whether it should have been possible to evolve good Tron players by self-play alone. This may be the case, perhaps a better algorithm for finding good agents than the one used in our coevolutionary experiments is conceivable.

Even with the highly sophisticated measure produced, which involves evaluating each agent against all agents selected form a 3-year, continuously running coevolution algorithm (i.e., the novelty engine), we still have a 11% uncertainty predicting the performance against people.

But this fact is only provable *a posteriori*, when the experiment has played thousands of games and measured hundreds of players along the way.

Graph 3.22 proves the success of our selection procedure; our system consistently performs better than the novice robots being produced by the novelty engine, by an approximate margin of 0.5 RS points, that is, by odds of 62%. Without selection against humans

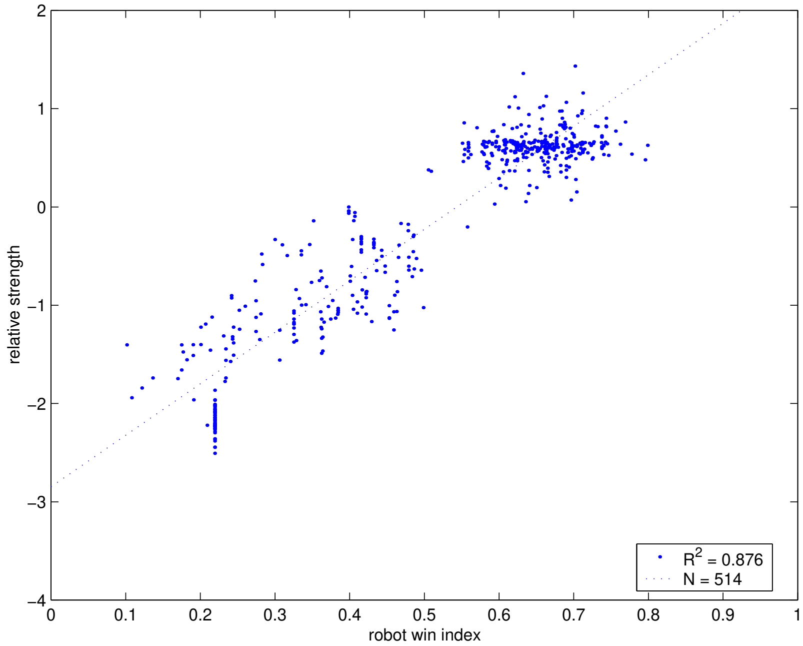


Figure 3.24: *Robotic fitness vs. RS*. All 5750 robots that have ever played humans were evaluated against themselves in a round-robin tournament (33 million games!) to extract the best self-play evaluation possible. We obtain a strong, but not total correlation (The broken line shows the linear regression) with the RS against humans (only robots who have played 100 games or more against people were considered).

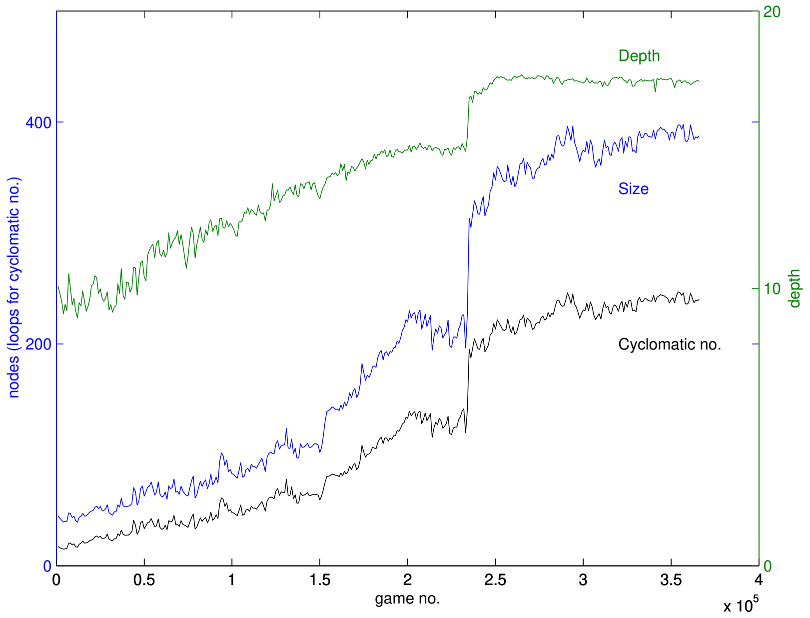


Figure 3.25: Evolution of Complexity in Tron Agents. The horizontal axis is the time scale. Three complexity measures are plotted: expression depth, size and cyclomatic number (depth is on the right-hand y axis). All three complexity measures increase with evolutionary time. Depth has reached an internal limit (17) whereas size could still grow (the maximum is 512).

the performance would have been weaker by at least this margin.

3.8 Emergent Behaviors

3.8.1 Standard Complexity Measures

Does selection favor more complex forms? The answer is yes. As indicated by figure 3.25, the complexity of the average Tron agent has been increasing over time.

We have taken three straightforward software complexity measures and applied them to the s-expressions governing Tron agents: depth (of the expression tree), size (number of tokens) and cyclomatic number (irreducible loops — see page 3). As time and selection go by, the average complexity of the surviving agents' code goes up. depth variable has reached a ceiling; most surviving agents have the maximum depth of 17.

But expression complexity is an abstract measure. Are we evolving complex *behaviors*? The present section analyzes the behavioral characteristics of evolved Tron robots.

3.8.2 Analysis of sample robots

Among the first robots moderately good vs. people was R.510006:

```
( * _H ( IFLTE _A 0.92063 _H ( - ( % D ( - ( + 0.92063 ( IFLTE 0.92063 _F
0.92063 ( LEFT_TURN ) ) ) ( IFLTE ( - ( IFLTE _C _G ( RIGHT_TURN ) (
LEFT_TURN ) ) ( IFLTE ( + ( LEFT_TURN ) ( LEFT_TURN ) ) _G _F _G ) ) _H
( RIGHT_TURN ) _G ) ) ) ( RIGHT_TURN ) ) ) )
```

This can be roughly reduced to pseudocode as:

```
if FRONT < 0.92063 go straight
else if 0.92063 >= REAR_RIGHT turn left
else if LEFT < RIGHT turn right
else turn left
```

This robot will always go straight, unless there is an obstacle in front of it closer than 8% of the size of the arena. At this point, it will turn right or left. The use of the REAR_RIGHT sensor is confusing, and its is difficult to infer from the code the actual behavior of this expression, as complex variations arise from its interactions with the Tron environment.

When inserted in a Tron arena, the agent shows an interesting behavior. It will avoid obstacles, get out of dead ends and do tight turns to maximize space when in a confined space (fig. 3.26).

The basic strategy of going straight all the time, then turning at the last minute, is one of the obvious solutions for reasonable Tron agents, given the established architecture. But robots can do much better than this: Today R. 510006 ranks at no. 2740 among 5977 robots, which means that 45% of the strategies found are better than its own.

3.8.3 An Advanced Agent

We now take a more detailed look at one of the top agents. R. 5210008 is at the time of this analysis, the top rated agent with an experience of more than 400 games; it has played 528 games and is the tenth ranked robot among those with 100 or more games played.

The s-expression of this agent has 449 tokens (see fig. 3.27). But even the cyclomatic number of this expression is 283: a very complex formula indeed. Software complexity measures [100] suggest that cyclomatic numbers higher than 10 should be "avoided" because they make a program difficult to understand. The cyclomatic number fails to capture some of the inherent complexity of the formula, such as the widespread use of templates such as **(- C _)** and **(IFLTE G _ _ _)**.

With its difficult structure, we were not able to manually de-compile the expression into pseudocode to try and follow the logic. The beginning of such pseudocode would be:

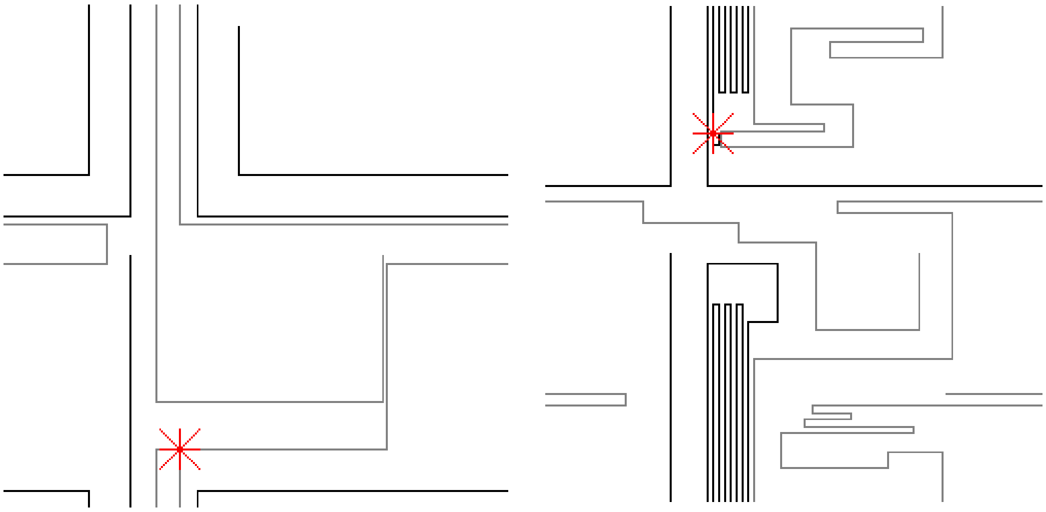


Figure 3.26: Sample games of robot 510006 (black) vs. a human opponent (grey). It quickly beats a novice player (left). It fights against an expert player, making tight turns when confined in a small space (right).

```
( IFLTE _G ( - ( - _A ( - ( IFLTE _C ( + _A ( IFLTE _E ( - ( - _F _A ) ( - _E ( - _C _G ) ) ) _A _D ) ) _C
_D ) ( - ( IFLTE ( - ( + ( + _C ( - ( * _C ( - ( - _E _H ) _C ) ) _A ) ) _A ) ( - _F ( - ( + _G _A ) _H ) ) ) ( -
_B ( + ( * _H _E ) _F ) ) ( - _C _A ) ( IFLTE _G ( - ( - _C ( IFLTE _G _A _F _F ) ) _C ) ( % ( IFLTE _A
( + _B _D ) ( - _C _C ) ( RIGHT_TURN ) ) _H ) ( IFLTE ( + ( IFLTE _E _C _C _D ) ( IFLTE ( IFLTE ( -
_A _C ) ( IFLTE ( IFLTE ( - _C ( - _F _A ) ) _D _A ( IFLTE _G _C _F _C ) ) _E ( + ( - _C _H ) _E ) (
RIGHT_TURN ) ) _A _D ) _A ( - _F ( - _C ( - _C ( + 0.06349 _A ) ) ) ) ( RIGHT_TURN ) ) ) ( - _A ( + (
* _F _B ) _G ) ) _A _A ) ) ) ( - _E ( - _A ( - _F ( - _C ( - _C ( + 0.06349 _F ) ) ) ) ) ) ) ) ) ( - ( IFLTE _G
_D ( IFLTE ( - _E ( - _E _A ) ) ( IFLTE ( - ( + ( + ( + _C ( - ( * _C ( + _C _F ) ) _H ) ) ( - ( * _E ( - ( - _E
_H ) _C ) ) _A ) ) ( IFLTE ( IFLTE ( * ( - _E ( - _A _C ) ) _C ) ( - _A ( + ( + ( - _C 0.06349 ) _B ) _C ) )
_E _C ) ( * _F _C ) _F _A ) ) ( - _F ( - _C _H ) ) ) ( - _B ( + ( * _H _E ) _F ) ) ( - _C _A ) ( IFLTE _A (
- ( - ( - ( * ( + ( - ( - _H _D ) 0.06349 ) _C ) _A ) ( IFLTE _A ( IFLTE ( IFLTE ( - _C _A ) _D ( - ( - _D
_A ) _D ) ( IFLTE _G _C _A _F ) ) ( + ( IFLTE _E _B _C _A ) _G ) _A ( RIGHT_TURN ) ) _G _H ) ) (
IFLTE _G _H _F _F ) ) _C ) ( % ( IFLTE _A ( + _B _A ) ( - _C _C ) ( RIGHT_TURN ) ) _H ) ( IFLTE (
+ ( IFLTE _E _C _C _D ) ( IFLTE ( IFLTE ( - _A _C ) ( IFLTE ( IFLTE ( - _C ( - _F _A ) ) _D _A ( IFLTE
_G _C ( + ( % _D _A ) ( LEFT_TURN ) ) _C ) ) ( + ( - _H ( - _A _C ) ) _A ) ( + ( - _C 0.06349 ) _E )
( RIGHT_TURN ) ) _A _D ) _A ( - _F ( - _C _C ) ) ( RIGHT_TURN ) ) ) ( IFLTE _D ( - ( + ( - ( + _G
_D ) _A ) _D ) _E ) _B ( - _A _A ) ) _C ( * _A ( * _H _E ) ) ) ) ) ) _B _E ) ( IFLTE ( + ( IFLTE _E _C _A
_D ) _G ) _C _C _H ) ) ( IFLTE _F ( - _C _G ) ( IFLTE _G _C _F _F ) _A ) ) ) ( - ( * ( * ( - ( * _C ( -
_C ( IFLTE ( - ( - _F _A ) ( * _F ( IFLTE _G _C _D _A ) ) ) ( * _D ( IFLTE _G _C ( + ( LEFT_TURN ) (
LEFT_TURN ) ) ( * _B ( RIGHT_TURN ) ) ) ) ) _F _A ) ) ) ( * _C _G ) ) ( IFLTE ( * ( IFLTE ( - _A ( - _F (
- _C ( - _C ( * _B _E ) ) ) ) ) ) _E _B _C ) ( IFLTE _H _C ( + ( LEFT_TURN ) _F ) _A ) ) _C _E _B ) )
_G ) _D ) ( + _F ( IFLTE _E _C _A _D ) ) )
```

Figure 3.27: The code (s-expression) of agent 5210008.

```
if (LEFT + RIGHT + RIGHT * (1 - RIGHT -
FRONT_LEFT)) > (FRONT_RIGHT - FRONT)
then
    If FRONT< 1 - FRONT_LEFT
    Then
        x = FRONT - RIGHT
    else
        x = REAR_RIGHT;
    if  x  > FRONT  turn right
    else y = FRONT
else
    y = RIGHT - FRONT;
if LEFT < REAR_RIGHT
then
    [...]
```

The first inequality involves 4 different sensors in a complex relationship, including a confusing multiplication and is already difficult to understand; it may generate a right turn or not, in which case the evaluation continues. And this is just the first expression of a long program, which would take several pages of pseudocode. Instead we have manually looked at several games from our historic records — to see how the agent behaves in the game environment.

So the next section, which talks about emergent behaviors in general, takes most of its examples from this player. We found out that it is capable of producing a surprising assortment of different behaviors including a different opening move for each game, cutting off the opponent , open box "traps" , driving around the edges and so on (figs. 3.31, 3.32, 3.33, 3.34 and 3.35).

3.8.4 Emergent Behaviors

Tron agents, by their architecture, are purely reactive entities — there is no state at all, no history or even access to the current position of the adversary. All behaviors are thus emergent, in the sense that they appear only as a result of the changing environment.

The architecture of Tron was designed in 1997, when many researchers in the adaptive behavior community were proposing basic *reactive* agents whose complex behavior relied more on the complexities of the environment than on complex reasoning and planning [14]. Consequently, Tron agents were deprived of any planning capability; they have no internal state, their sensory inputs are very restricted and all the complex behaviors that we observe are the result of their situatedness: Tron agents behave in complex ways as a results of their being adapted to a complex environment over a long evolutionary process.

Placed on a fixed environment, a Tron agent would have a constant behavior, either right or left turn, or straight. This of course, never happens: even in the absence of an opponent, a Tron agent is constantly moving, generating a trail that immediately becomes part of the environment; such changes will be perceived and thus different actions begin to occur as a result of the re-evaluation of the agent's control expression.

Among the information sent back by the Tron Java applets, and stored in our server's database, is the full trace of game. All turns are recorded, so we can re-enact and study the games that took place between humans and agents.

From this record we have selected a few snapshots that showcase Tron agents performing different high-level behaviors during their games with humans. Every snapshot is labelled with the time[7] at which the game finished, the id number of the human opponent and the robot's id number. An asterisk marks the losing player (or both in the case of a tie).

The trace of the robot player is a black line, the human a grey line.

Spiral inwards Tracing a box and then spiraling inwards is a commonplace defensive behavior (fig. 3.28). Once the box is marked, the opponent cannot get in; a spiral-in then exploits this gained territory optimally.

Live and let live Sometimes two players avoid each other, staying away from confrontation, for as long as possible. Two commonplace occurrences in agent-vs-agent coevolutionary scenarios are loops across the vertical dimension, as seen on fig. 3.29 (left). Another strategy that attempts to stay away from the opponent is to spiral outwards from the starting position (fig. 3.29, right).

We think that this phenomenon occurs as a form of cooperation between coevolving agents: agents that agree on this behavior split the point with each other, engaging in a form of cooperation: any deviation from this behavior would be retaliated against. Figure 3.3 depicts two robots that persist on their behavior until they crash simultaneously, thus producing a tied game.

We were surprised to observe humans engaging in the same type of strategy. In these two examples we see the human imitating the agent's behavior — albeit imperfectly. On the first example, H. 457 died first; but on the second, H. 259 successfully broke the pattern, and R. 370002 lost the match.

Staircasing Only orthogonal movement is defined by the rules of Tron. One can move in a horizontal or vertical direction, never diagonal. But Tron agents often simulate a diagonal,

[7] In UNIX-style time units (number of seconds after 0:00 01/01/1970).

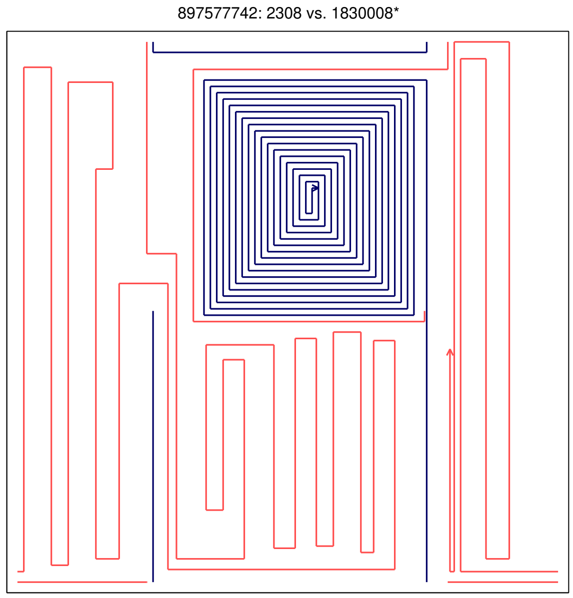


Figure 3.28: *Spiral Inwards.* A spiral inwards is a good solution to make the most out of a closed space (black=agent, gray=human).

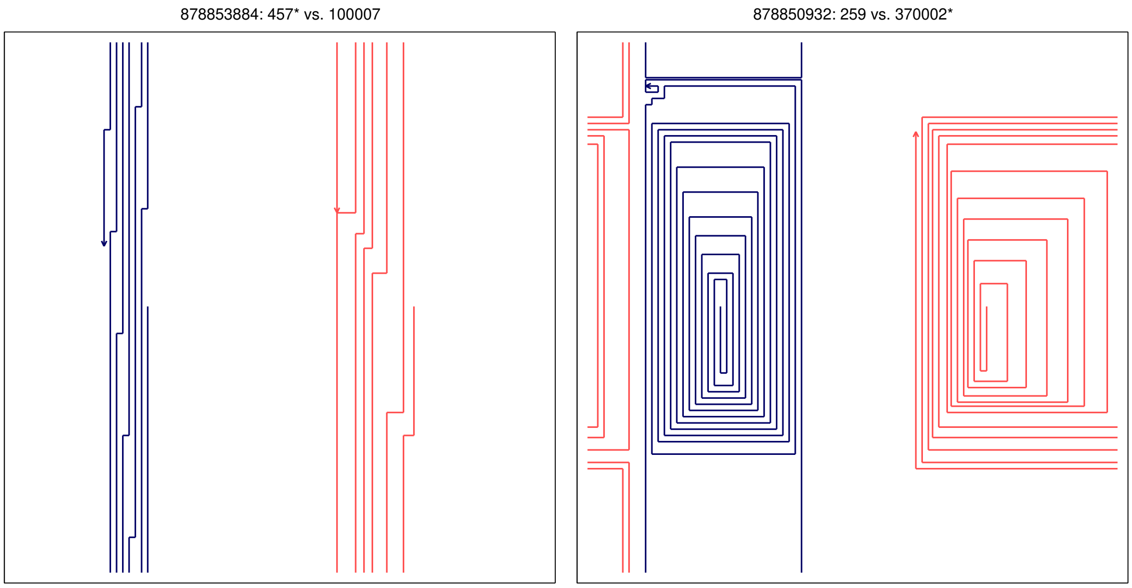


Figure 3.29: *Live and Let Live.* In both these games, human and agent agree on a common strategy to avoid confrontation. This often occurs as a form of emergent collusion in coevolutionary games, yet it's surprising to observe it occur between human and agent (black=agent, gray=human).

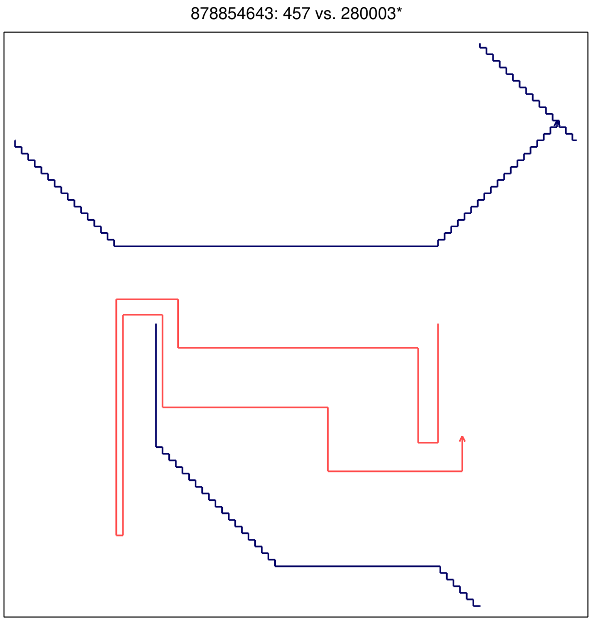


Figure 3.30: *Too much staircasing.* Agent 280003 seems to know about staircasing and little else (black=agent, gray=human).

by quickly alternating left and right turns. It may be a bit disconcerting for the human but, by itself, is not enough to be a good player (fig. 3.30).

Opening moves The first few seconds of a match set up the type of game that will be played. A player has the option to try and stay away from its opponent, or go after him; to stay close to the starting position or mark territory away from it. In any case, performing the same opening moves in every game seems like a bad idea, for opponents would be able to adapt to it. Performing a different opening move every time is difficult to accomplish for deterministic agents that start every game from the same configuration. Figure 3.31, however, shows a series of games played by R. 5210008 (only the first 300 steps of every game). All these games have different initial moves.

The cutoff maneuver A common move among humans is the "cutoff": If you and your opponent are running parallel to each other, but you are ahead, cut him off and try to beat his reaction. One wouldn't expect to see an agent perform the cutoff — they don't know where you are! Figure 3.32 however, depicts R. 5210008 performing what looks like a typical cutoff.

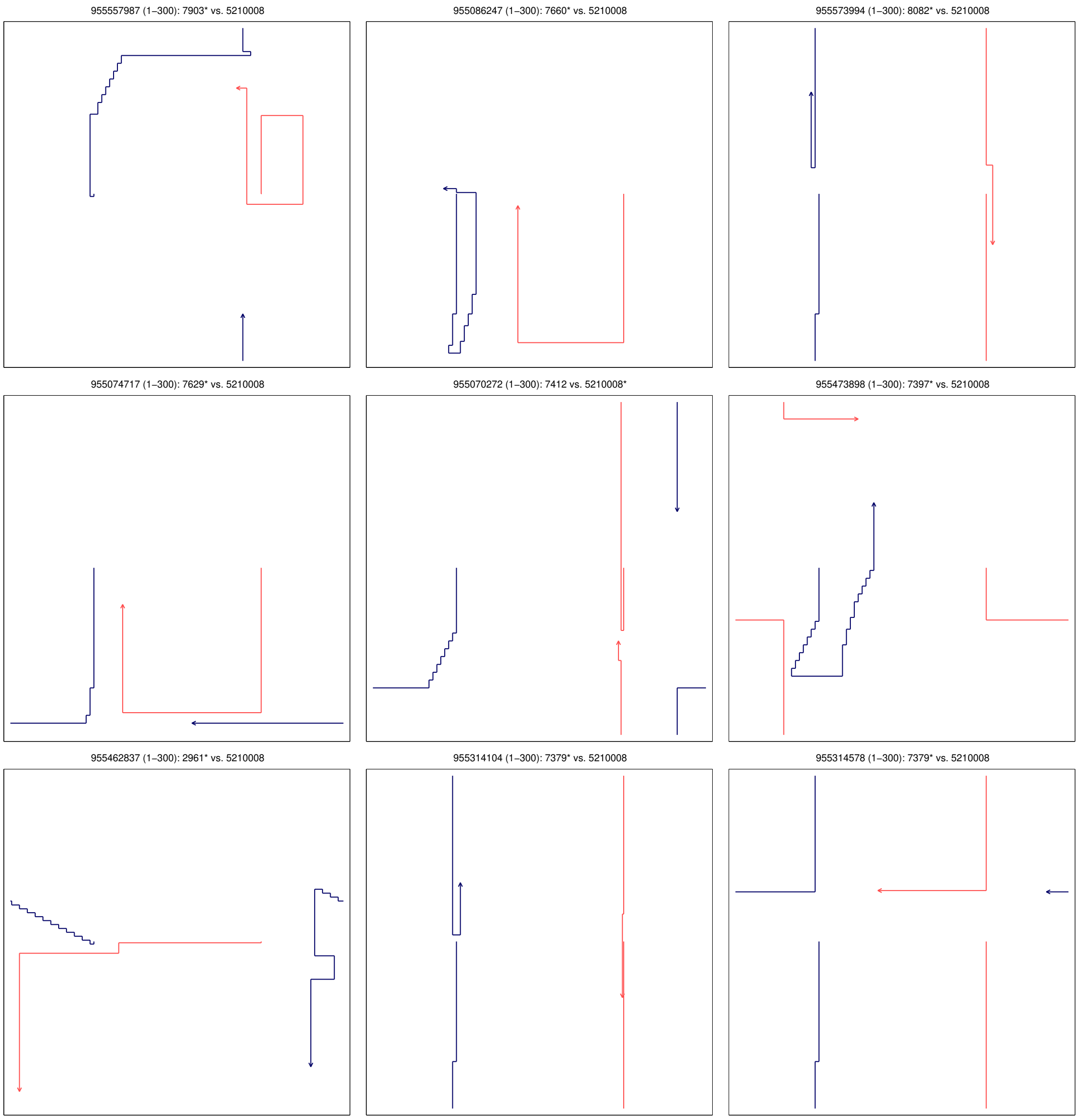


Figure 3.31: *Changing Behavior.* Agent 5210008 showcases here its variety of opening moves (black=agent, gray=human).

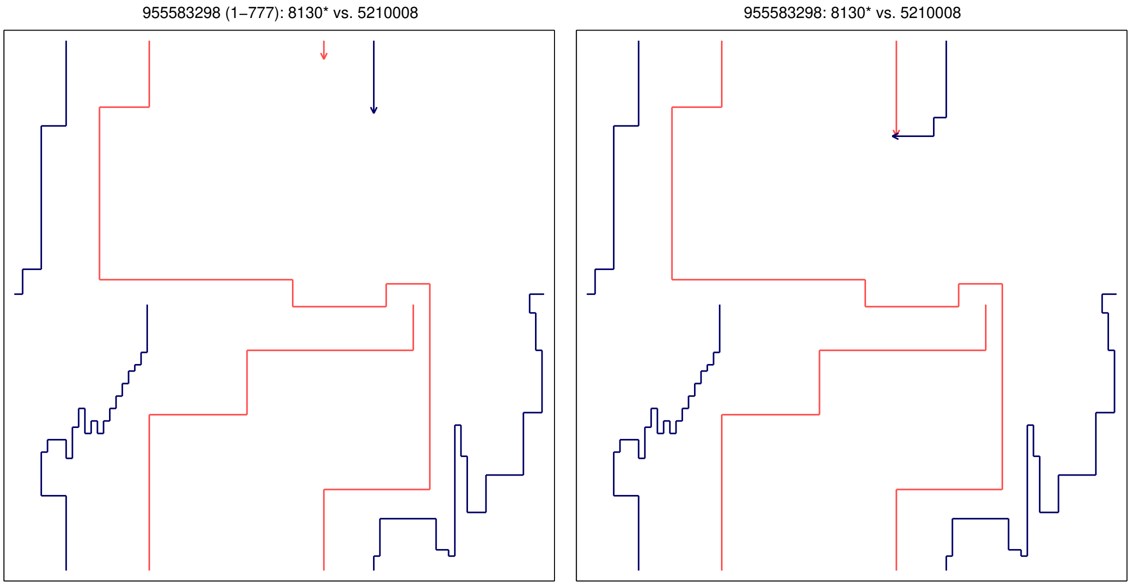


Figure 3.32: *Cutoff. R.* 5210008 is seen here performing the "cutoff" maneuver. Human and agent find themselves running parallel (left), but the agent is ahead, so it makes a sharp right turn, cutting off the human (black=agent, gray=human).

Topological games During the game depicted on fig. 3.33, agent 5210008 created a box with a small opening. This box spans across the screen boundary, so it is not easy to perceive for a human player. When the opponent finds himself in the box, it may be difficult to go back and find the narrow exit.

Combined behaviors Fig. 3.34 shows an agent combining different behaviors: wandering, wall following, obstacle avoidance, space-filling turns, to win a game.

Edging Running along the edges of the screen is a strategy that easily creates confusion in humans. A human player approaching the edge often finds it difficult to quickly look at the other edge to plan ahead for what is going after emerging on the other side. A wall along it is difficult to see. Agents on the other hand, are immune to this strategy — the border of the screen is invisible to them; their sensors have no perception of borders, they don't really exist, being an artifact of how the display was encoded for visual human input (fig. 3.35).

Space Filling It often happens that an end game is reached where both opponents are confined in a space, isolated from the opponent. At this point the optimal strategy becomes filling the space as densely as possible, hoping for the other opponent to crash sooner. Any

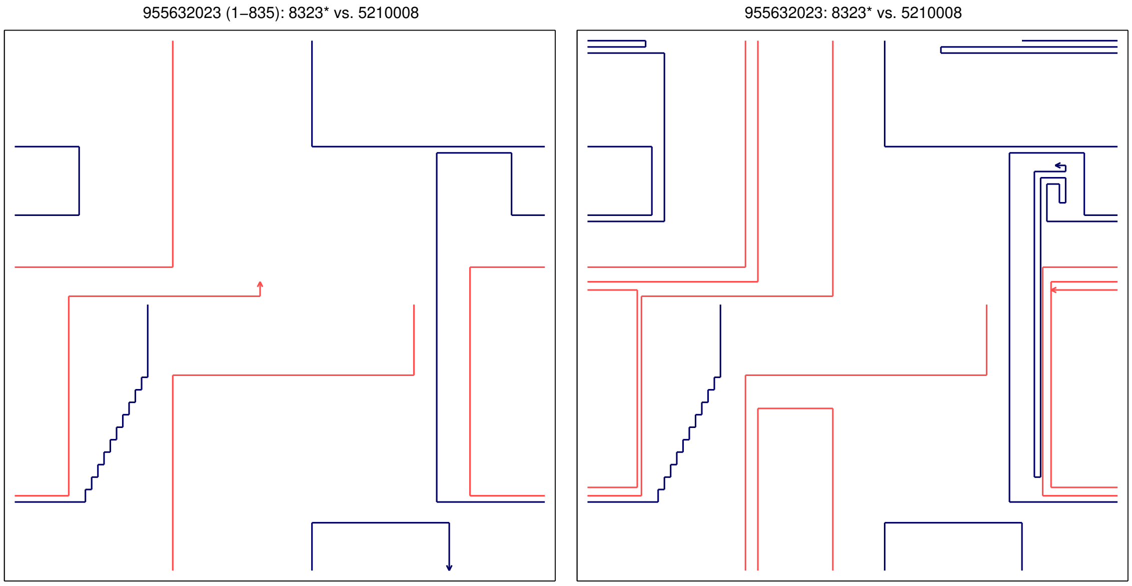


Figure 3.33: *Trapped*. R. 5210008 created a box at the edge of the screen, with a small opening, and its opponent fell inside it. After visiting the trap once, H. 8323 goes up (left), ultimately landing in the trap again. Eventually the human is boxed in and commits suicide (right).

good Tron player should master this technique. The game depicted on fig. 3.36 shows human and agent in a match of endurance.

Unforced errors Even the highest ranked Tron agents seem to make an occasional mistake, crashing against themselves in a silly manner for example (fig. 3.37). People do the same, even the best player makes an occasional mistake.

Behavior or Imagination?

Are these behaviors really occurring as we described them? Some cases, such as spiraling, are quite obvious, but others like the *cutoff* and the *trap* might be more in the imagination of the observer. After all, a Tron agent cannot decide that the time is ripe for a cutoff maneuver without knowing the position of the other player. Nor it has memory to decide: "I am in the process of doing a spiral, so turn in the same direction again".

On the other hand, it might be the case that the cutoff is "definable" for an agent in different terms than ours. As long as there is a certain environmental condition (in terms of sensory inputs) that is correlated with the right moment to do a cutoff, agents likely to turn in the correct direction in those circumstances might have an evolutionary advantage, so in the end we see agents "doing it" with increasing frequency.

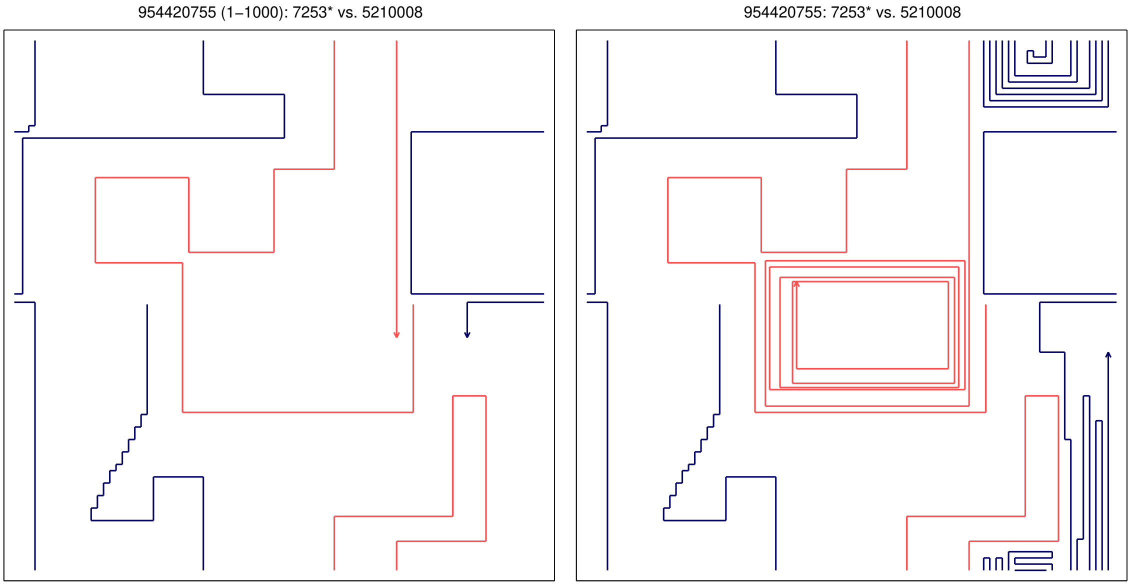


Figure 3.34: *Emergent Behaviors* R. 5210008 displays different emergent behaviors in this match. In the first part of the game it uses a combination of wandering and obstacle avoidance strategies, cutting off the arena in small spaces (left). As the game progresses, a combination of tight turns and wall following help make the most out of a more confined situation (right) (black=agent, gray=human).

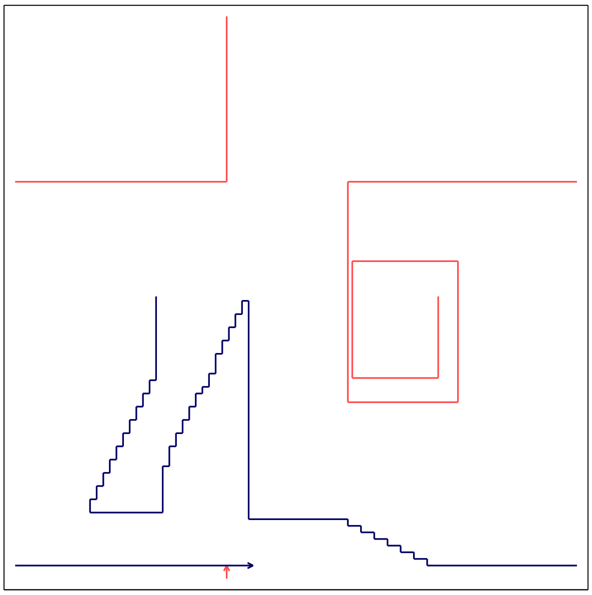


Figure 3.35: *Edging and Staircasing*. Here R. 5210008 shows the use of staircasing to produce a diagonal displacement in a domain where only horizontal and vertical movements were originally defined. Creating walls along the edges of the screen, as in this game, allows Tron agents to exploit one human handicap: humans have trouble understanding the toroidal topology where edges are connected. In this game the human attempted to go across the border and crashed the wall that the agent had built (black=agent, gray=human).

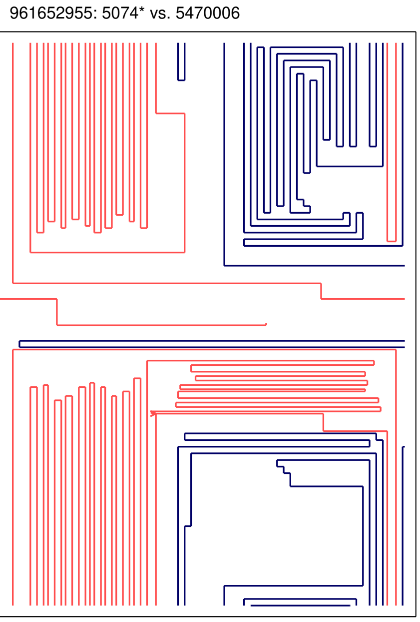


Figure 3.36: *Space Filling.* R. 5470006 and Human 5074 engage in a space-filling endurance contest. The difference between their perceptual modes is visible: The human is allowed tighter turns, yet its finger ability is limited. The agent cannot measure with precision the space remaining, so sometimes recurs to spiraling in, then out as in the upper right region. In the end the human fails to enter a narrow passage and loses (black=agent, gray=human).

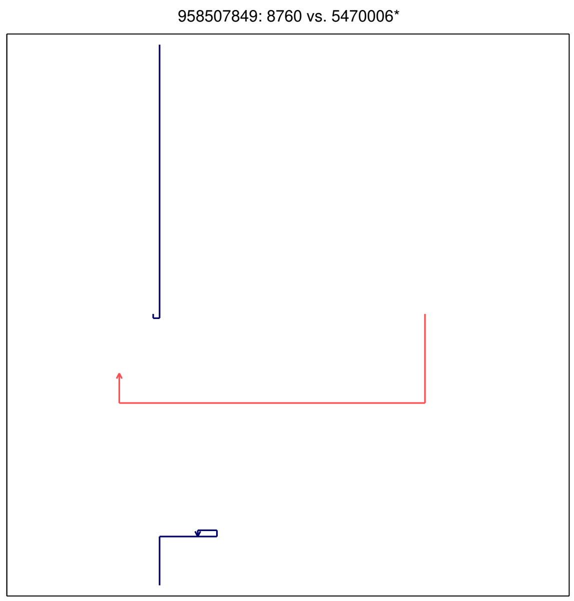


Figure 3.37: *Error.* R. 5470006, one of the all-time best Tron agents, loses here in a stupid manner, crashing itself. Even though this is a highly evolved agent capable of winning against most human opponents, it still makes occasional mistakes (black=agent, gray=human).

Ethologists warn us [106, p. 105] that an animal's action is called *behavior* only when performed in order to improve its chances of survival or reproduction — because it has been selected for, that is. The present section just shows a group of snapshots to illustrate the types of actions that Tron agents are performing. In the next one we go further, testing for correlations between behaviors, evolutionary time, and fitness.

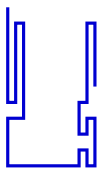
3.8.5 Quantitative Analysis of Behaviors

Emergent behaviors are difficult to quantify: how could we measure "wandering" ? In this section we examine some behaviors that we were able to define with simple formulas, allowing us to count their occurrences quickly by using regular expressions.

Two consecutive turns of a Tron player can be either in the same direction (left-left or right-right) or in opposite directions (right-left or left-right). The former are *U* moves, and the latter *S* moves. The *size* of the move is the number of pixels advanced between the two turns. Most of the different behaviors we quantified are formally defined by a regular expression on the sequence of turns thus defined (table 3.4).

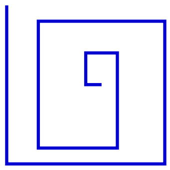
Behavior	Expression
Tight Turn	$U < 6$
Spiral	U^3
Staircase	S^3
Zig-zag	USU
Loop	$(\{U\|S]\} > 150)(S < 14)$
Diagonal	$(S < 15)^3$
Zig-zag Fill	$(U < 6)S(U < 6)$
Turns	*turns*
Asymmetry	$\left\| \frac{left\ turns - right\ turns}{turns} \right\|$
Edge Crossing	*distance to edge = 0*
Edging	*distance to edge* $\leq$*10* and *parallel to edge*
Spiral In & Out	U^3SU^3

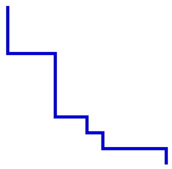
Table 3.3: Definitions of Behaviors



1. Tight Turns A tight turn is defined as two consecutive turns to the same side, within 6 steps of each other ($U < 6$). The icon depicts a player doing 8 tight U turns.

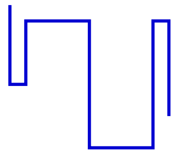
2. Spirals A spiral can be outwards or inwards; it is characterized as making four turns in the same direction.

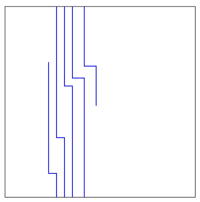




3. Staircase A staircase is defined as 4 consecutive alternating turns.

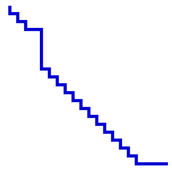
4. Zig-zag A player is "zig-zagging" when alternating left-left then right-right turns.

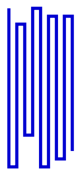




5. Looping A player "loops" around the torus when she goes straight for more than 150 steps, then makes a quick 'S' turn (less than 14 steps) to make a second pass parallel to the first one (fig. 3.29 left).

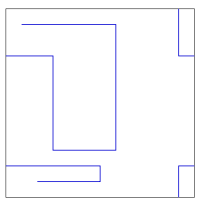
6. Diagonal A diagonal move is a tight staircase, all turns within 15 steps of each other (fig. 3.30).





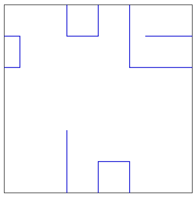
7. Zig-zag filling This behavior is a combination of zig-zag and tight turns; a succession of 2 opposite tight U turns. It is a useful strategy to fill up the space tightly. The human player on fig. 3.36 spent most of the match executing this behavior.

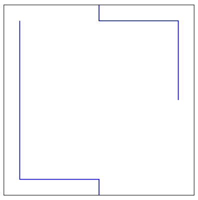
8. Turns Some players go straight for a long time, others are turning all the time. This feature could also be called "nervousness".



9. Asymmetry Some players prefer to do mostly left turns or right turns; others have a 50%-50% balance. According to the definition on table 3.3, an asymmetry equal to one means all turns have been made to the same side, and a zero asymmetry means an equal number of turns to both sides. The icon depicts a player making seven right turns but just one left turn (6/8 asymmetry = 0.75).

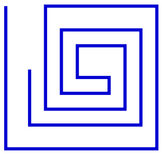
10. Edge Crossing Each time a player crosses the edge of the screen, to reappear on the opposite side. Agents do this without noticing, since they have no perception of the arena having edges; they just see a continuous topology. Humans need to learn to mentally connect the opposite borders of the screen. The icon depicts a player going across the edge five times.





11. Edging Each step a player runs parallel to the edge of the screen within 10 pixels of it (see fig. 3.35).

12. Spiral in and Out We have observed that one of the ways Tron agents may use to gain a space and exploit it, is a spiral that goes in first, then out. This player can spiral in loosely, then go backwards, spiraling out. The pure spiral behavior often ends in a crash (fig. 3.28) but after spiraling in, then out, the player is still alive. The agent of fig. 3.36 used this strategy at the upper right corner of the arena. We have defined it as four consecutive turns to one side (spiral) followed by four turns to the opposite side (opposite spiral).



The first question we wish to quantify is: Are any of these simple behaviors being favored or disfavored by evolution? Is our system consistently relying on any of them? The first group of results, fig. 3.38, shows the frequency of behaviors along the history of the system. We grouped all games in bins of 1000, and counted the occurrences of each of the 12 patterns, divided by the number of steps. So each behavior is quantified as the "propensity" of the Tron system to engage in it (except for *asymmetry* which is a per-turn ratio, not per-step).

The behaviors that occur with increased frequency along time are tight turns, zig-zags, filling zig-zags, edge following and spiral-in-out. Decreasing in frequency are spiraling and asymmetry. The others do not show a clear tendency. This confirms some of our intuitions, such as filling spaces and edging being favored by evolution. Spiraling on the other hand, is surprisingly disfavored. We must conclude that spiraling is a bad idea in general, albeit a necessary tool for a behavior that is a specialization of spiral — spiral-in-then-out, which is favored by evolution.

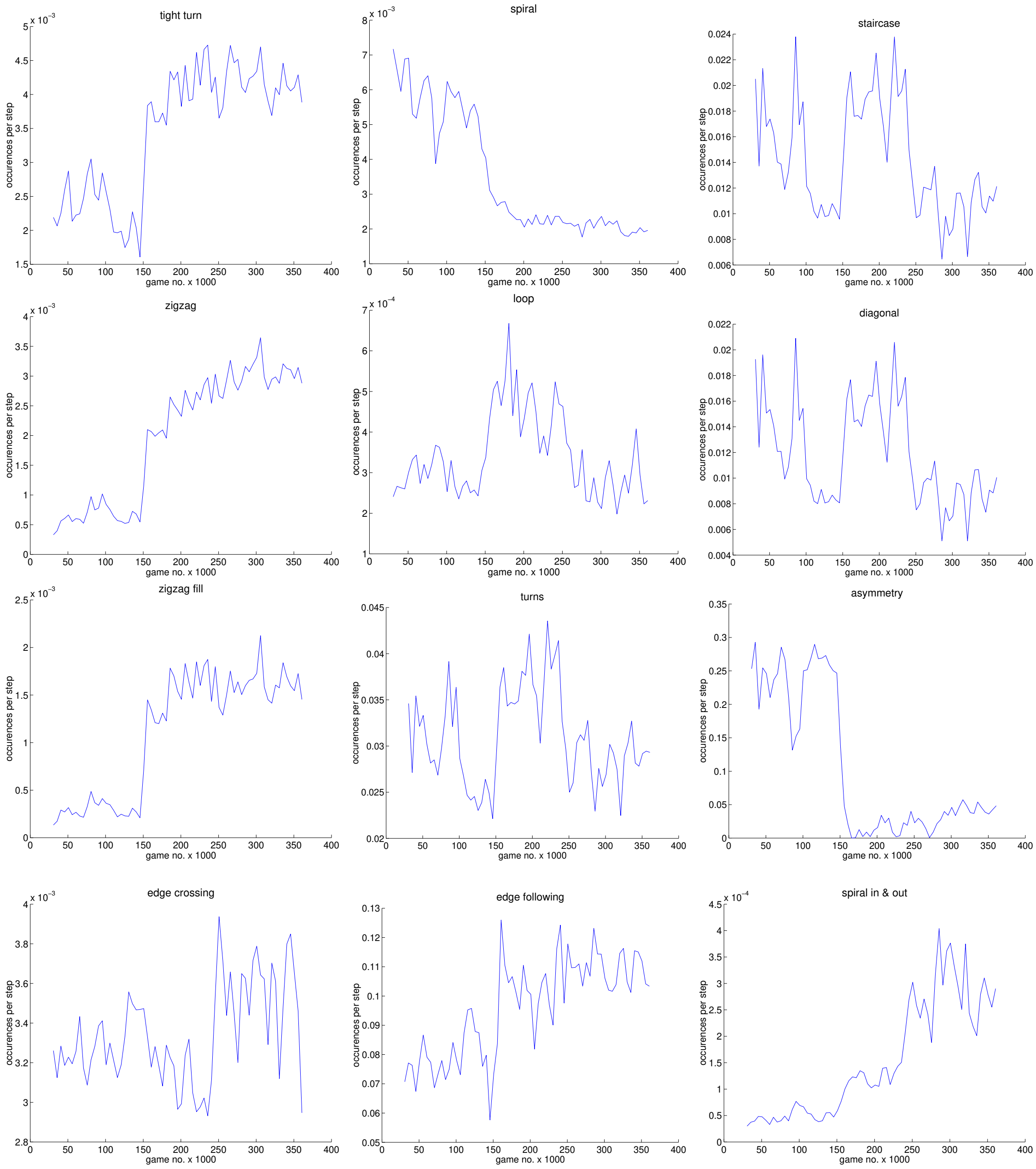


Figure 3.38: *Behaviors vs. Time*. U turns, spirals, staircasing, zigzag, loop, diagonal, zigzag-fill, turns, asymmetry, edge crossing, edge following, spiral in, then out. Games were sampled in groups of 1000 and average events of each behavior, per game step, plotted on the vertical axis.

The next question is, are these behaviors associated with a robot's measured strength? Do some behaviors occur more or less often in stronger robots? The associations should not be much different from those defined by time, since performance is being selected across the time dimension. To observe this, we have selected all robots whose accumulated games are at least 10000 steps, and measured their behavioral frequency. On fig. 3.40 we have marked one point per robot; the x coordinate being the strength, the y coordinate the behavioral frequency. The result is a cloud of points, quite disperse in all cases. This means that none of these behaviors implies, by itself, that an agent is strong or weak. Quite the opposite, for each behavior one can usually find both good and bad players who perform it either often or rarely.

The clouds of points do show some accumulation regions though, so we also divided the RS axis in six segments and calculated the mean and its standard error. We confirm with this that, on average, better robots are doing more tight turns, zig-zags, filling zig-zags and spirals in/out but less spirals — and tend to have symmetrical behaviors and follow the edges of the screen.

3.8.6 Differences Between Human and Agent Behaviors

In this section we analyze the differences between agent and human behavior, according to the 12 "quantifiable" behaviors described on the previous section. Figure 3.40 shows the results. The graphs in this figure repeat the curves for robot behavior frequency vs. performance (same as in 3.39), adding the curves for the human case. .

Table 3.4 summarizes these results, comparing four categories: novice agents, advanced agents, novice humans and advanced humans.

These are the differences for each individual behavior:

Tight turns Agents develop the capacity for doing tight turns early on. Due to their error-free sensors, they can perform this type of maneuver very efficiently. The more advanced the agent, the more frequent the behavior becomes.

For humans, doing closed turns with exact precision requires training. As with agents, there is a strong correlation between frequency of tight turns and performance. A top human, on average, performs tight turns as often as a beginner agent.

Spiral Although it does happen sometimes (see fig. 3.29), this is not a frequent behavior for people. The inwards spiral amounts to creating a confined space and entering it, something that human biases warn us against. The outwards spiral also seems pointless, it is a passive behavior that neither attacks nor runs away from the attacker.

The opposite is true for agents. Robots develop this strategy in the beginning of robot-robot coevolution scenarios, when most other strategies are random (hence

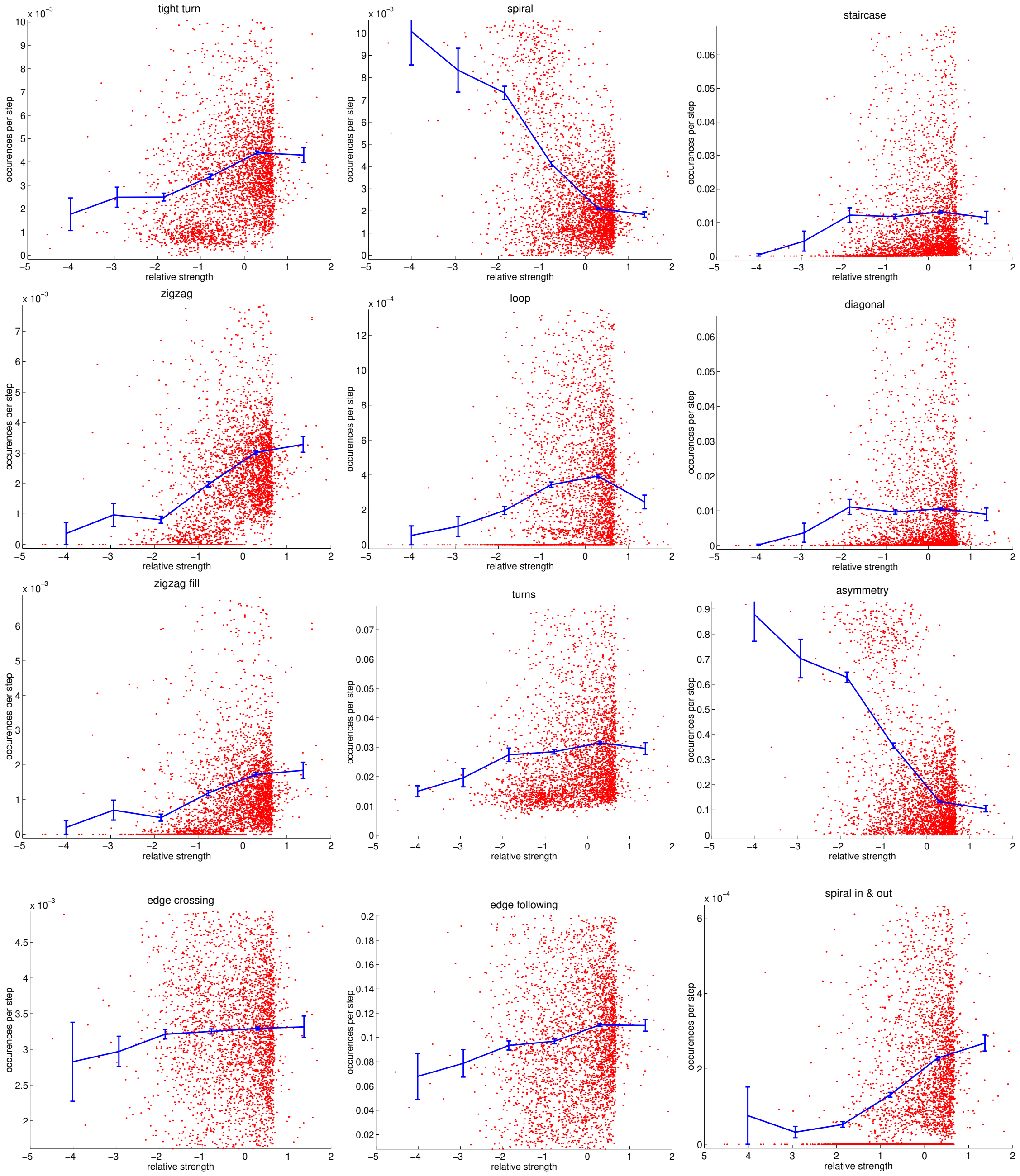


Figure 3.39: *Behaviors vs. Performance:* U turns, spirals, staircasing, zigzag, loop, diagonal, zigzag-fill, turns, asymmetry, edge crossing, edge following, spiral in, then out. Horizontal axis: robot RS; vertical axis: average events per game step. Every robot is plotted as one point. The broken line is a histogram showing mean occurrences along 6 intervals on the RS range. Error bars mark the standard error of the mean.

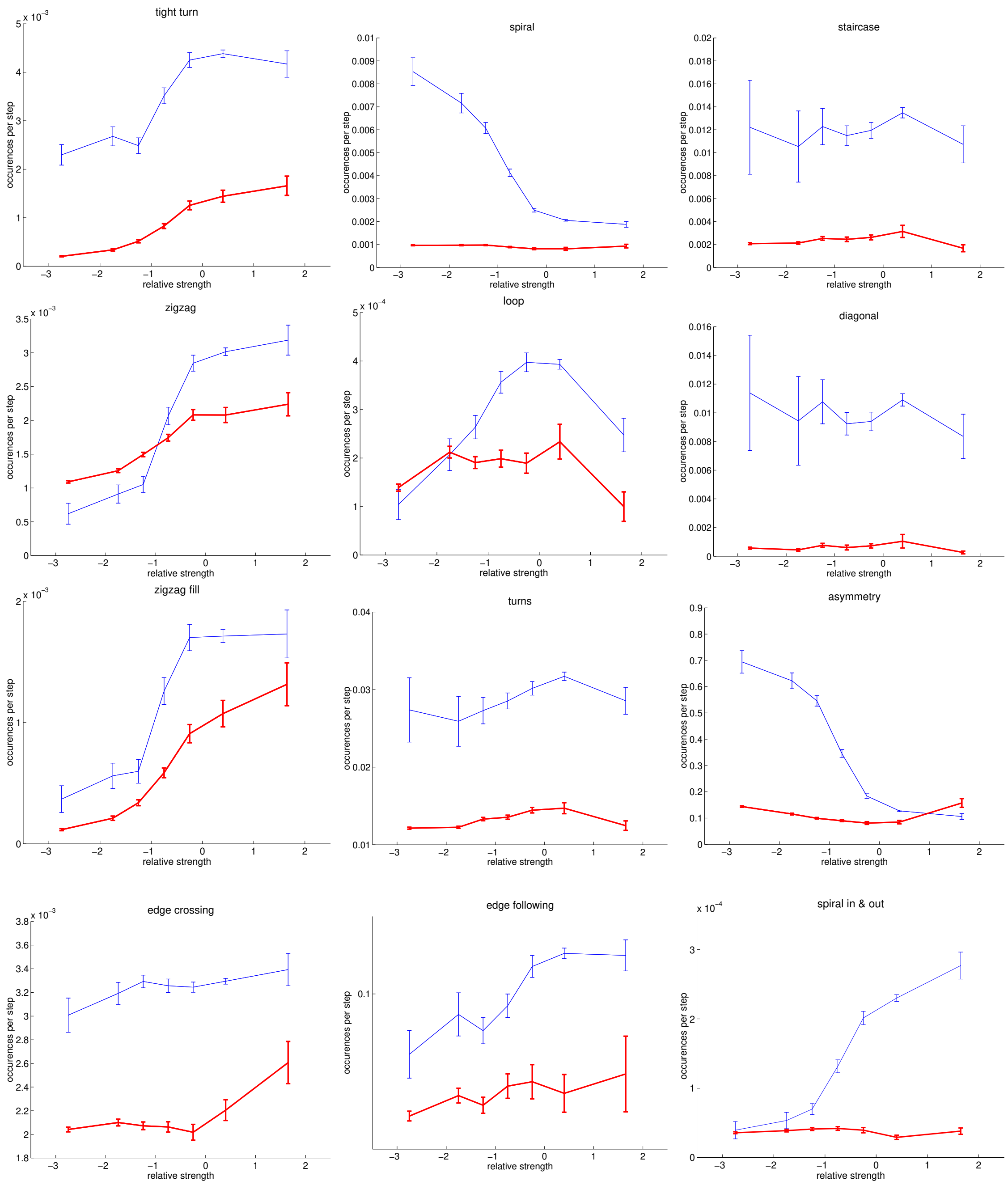


Figure 3.40: *Agent & Humans behavior frequencies vs. strength:* There are significant behavioral differences between agent (thin lines) and human (thick lines) behaviors, according to our 12 test cases. Horizontal axis: RS; vertical axis: average events per game step. Error bars indicate the standard error of the mean. The first and last bins are wider, to compensate for the sparsity of players at both ends of the performance scale.

from	Novice Agent			Advanced Agent		Novice Human
to	Advanced Agent	Novice Human	Advanced Human	Novice Human	Advanced Human	Advanced Human
tight turns	+	-	=	-	-	+
spiral	-	-	-	-	=	=
staircase	=	-	-	-	-	=
zigzag	+	=	+	-	-	+
loop	+	=	-	-	-	=
diagonal	=	-	-	-	-	=
zigzag fill	=	=	+	-	=	+
turns	=	-	-	-	-	=
asymmetry	-	-	-	=	=	=
edge crossing	=	-	-	-	-	+
edge following	+	-	=	-	-	=
spiral in&out	+	=	-	-	-	=

Table 3.4: *Correlations between humans, agents and behaviors*. Each column represents a pair of categories. The "=" symbol means that there is no big difference between both categories on the respective behavior, whereas "+" means that the second group has an increased value with respect to the second (and "-" the opposite). The first and last columns compare novices with advanced players, amongst agents and humans respectively. Tight turns for example, increase with level of play for both agents and humans (+); a novice agent doing about as many of them as an advanced human. Asymmetry is negatively correlated with quality (for robots) but uncorrelated for humans.

Agent Id.	Len	Code
230007	5	(IFLTE 0.88889 _C _C (LEFT_TURN))
230009	5	(IFLTE 0.88889 _C _C (LEFT_TURN))
230010	5	(IFLTE 0.88889 _C _C (LEFT_TURN))
90001	5	(IFLTE _D _C (LEFT_TURN) _D)
90002	5	(IFLTE _D _C (LEFT_TURN) _D)
510003	7	(* _H (IFLTE _A 0.90476 _H (RIGHT_TURN)))
50008	9	(* _H (IFLTE _A 0.90476 _H (RIGHT_TURN)))
60001	9	(IFLTE _B _F (IFLTE _C _H (LEFT_TURN) 0.11111) _F)
60002	9	(IFLTE _B _F (IFLTE _C _H (LEFT_TURN) 0.11111) _F)
60003	9	(IFLTE _B _F (IFLTE _C _H (LEFT_TURN) 0.11111) 0.12698)

Table 3.5: The shortest agents produced by the novelty engine have 5 tokens each. Agents 230007, 230009 and 230010 do a tight spiral. 90001 and 90002, a wide spiral (fig. 3.41). 510003 does something different: it goes straight until it reaches an obstacle. 60001-60003 do a sort of "Tit-for-tat"; they spiral while the other player is also spiraling, but break the pattern when the other player does so.

suicidal). Sometimes a whole population may fall into a *mediocre stable-state* [108] characterized by most agents doing spirals. The spiral is probably the simplest non-suicidal behavior in terms of GP code.

A search for the shortest robots ever produced by the novelty engine (table 3.5) reveals two minimal behaviors which use just 5 tokens. One of them, R230007 does a classic tight spiral, and the other, R. 90001, a more loose spiral. The code for R. 230007 is:

```
(IFLTE 0.88889 _C _C (LEFT_TURN))
```

which translates as:In the end, humans get out of mazes for the exact same reason.

```
if LEFT < 0.8888 then go straight else turn left
```

so this robot executes a left turn whenever there are no obstacles to the left. This minimal code results in an basic wall following that produces a tight spiral as depicted on fig. 3.41 (top). When the robot is running along its own wall, built by the previous lap, the left sensor perceives the obstacle and the agent goes straight. But as soon as the corner is reached, the space suddenly opens to the left and the agent turns.

As evolution progresses, agents "unlearn" to do spirals, finding better strategies. The behavior frequency diminishes sharply for more advanced agents, approaching the human average rate: In the best robots, spiraling has been almost completely abandoned.

Staircase Together with its tight version, the **diagonal**, staircasing is a characteristic behavior that strongly differentiates human and robotic playing styles. Agents perform a diagonal on 1% of their total game time on average, whereas the rate for humans is much lower, close to 0.05%.

A human's attention typically shifts between two modes: it either focuses on a narrow region around the present position, in order to perform precise maneuvers and turns, or spreads over a wider region, analyzing the different parts of the arena in an effort to plan the next move.

A move such as the staircase can be performed only in the narrow attention mode. When one switches to the second, "big picture" mode of attention, turns stop completely. So humans in general will not perform continuous turns for long periods of time.

Agents, on the other hand, lack attention characteristics altogether, so they can afford to be constantly turning without confusing or delaying their sensors readings or analysis.

Zigzag/Zigzag fill This is a behavior that shares similar frequency profiles for both species. Zigzagging is an important ability for the endgame, so its frequency increases with expertise on agents as well as on humans. The sample game shown on figure 3.36 illustrates how both species resort to zigzagging in similar situations.

The "filling" zigzag serves the purpose of making the most out of a confined space and amounts to about half of all zig-zags, in humans and robots alike. The frequency of filling zig-zag, for humans as well as agents, is an order of magnitude larger for expert players as compared to novices.

Loop Looping, together with spiraling and tight zigzagging, is a space-filling strategy (fig. 3.29, left). The correlations of looping and strength are unique, though: both humans and agents seem to increase looping with expertise, but only up to a certain point. In the end, the most expert players, humans or robots alike, have abandoned this behavior, frequencies falling down to beginners' levels.

Turns Another behavior that strongly differentiates humans and agents: agents are much more "nervous", they make turns more frequently. Robots turn once every 33 steps on average, whereas humans do so only once every 80 steps. Again we think that this difference is related to human attention modes, as in the staircase above.

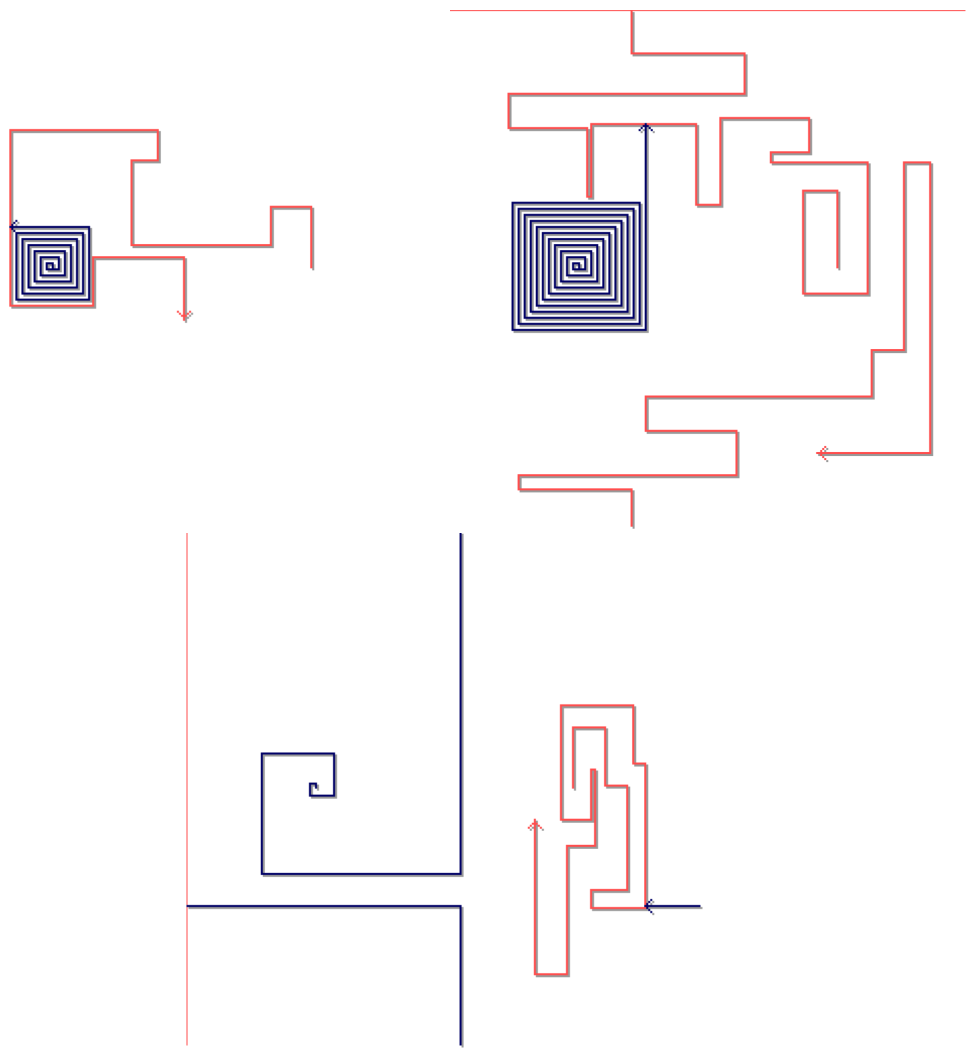


Figure 3.41: *Simplest Agent.* Sample games of the simplest agents according to code size (table 3.5). R. 230007 and R. 90001 are 5 tokens long. Agent 230007 does a tight spiral by means of a simple wall following, oblivious to what its opponent is doing (top left). This agent can sometimes break the spiral when it finds an obstacle (top right), by "following" the wall of an obstacle. The spiral of agent 90001 (bottom), created by comparing the left and rear-left sensors, is a Fibonacci spiral (the length of each segment equals the sum of the previous two).

Asymmetry Humans rarely depict any strong preference for turning to either side, whereas this is a typical characteristic of unsophisticated robots.

The reasons for asymmetric behavior on robots are similar to those explained for spiraling above: early on coevolutionary runs, a useful turn is discovered and exploited. The code will spread along the population and everybody will start performing the same type of turn. Later on, more advanced turning patterns are discovered that involve left as well as right turns.

In the end, the best agent strategies have perfectly balanced frequencies of left and right turns: levels of asymmetry are near-zero for advanced robots, and for humans of all levels.

Edge Crossing Unsurprisingly, robots cross the edges of the screen more often than humans. Robots do not perceive edges in any direct manner, so they move across without a problem.

Agents go across edges once every 300 game steps (approximately), whereas the human frequency is closer to one crossing every 500 game steps (a random walk would go across an edge every 256 steps).

Edge Following Another differentiating behavior, robots move close and parallel to the edges of the visible screen (at a distance of 10 or less, see fig. 3.35) more often than humans. Also, the percentage of game time they spend doing this, increases with the expertise of the agent.

A random walk would move along the edges 7.8% of the time. This is about the frequency for novice robots, but expert ones ‘edge’ about 12% of the time. Human rates stay between 2.5% and 5%, increasing slightly for experts.

Even though agents do not perceive edges — and thus are incapable of defining “edging” explicitly — the better ones do it more often than random. Thus, albeit indirectly defined, agents seem to have found a way to exploit a human weakness.

For humans, being close to an edge is perceived as dangerous: something might come up unexpectedly from the other side, so humans stay away from edges more often than not.

Spiral In & Out A behavior that occurs only amongst advanced robots. Difficult for humans, because it needs very precise navigation, robots discovered it at some point and now is a resource strongly correlated with better performance.

Altogether, we have found that the set of behaviors we have been analyzing has provided us with interesting measures of robot and human evolution and learning. Some of them are typical of the "robot" species: more tight turns, more crossings of the screen's edges, diagonals produced by quickly alternating turns.

Zigzag is a unique problem in that it seems about equally important, and equally difficult for agents and humans alike. Zigzagging is fundamental for split endgames, when both players are trying to save space, waiting for the other to make a mistake.

Some behaviors occur mostly at specific levels of expertise: Spiraling and asymmetry are typical of novice agents, whereas in-out spirals and edge following are characteristic behaviors of advanced agents. Among humans, tight turns and edge crossings are common tools of expert players.

None of these behaviors had more frequency on humans than robots. Perhaps our choice of 12 sample behaviors was biased by our observations of how agents behave, rather than humans. But it is also interesting to reflect on the fact that human behavior is more complex, more changing, so it is difficult to find fixed patterns that occur very often. Several behaviors have much larger frequencies amongst agents than humans: staircase, edge following, and frequency of turns.

This last characteristic, lower human frequency of turns, we conjecture is related to a fundamental difference on the way that agents and humans approach the game. Agents are reactive, they read their sensors and act immediately. Humans switch between different attention modes: they exploit safe situations, where they can go straight for a while without interruptions, to look at the opponent's behavior, examine remote areas of the board, study the current topology of the game situation, and make plans for the future. Even though strategically it makes no difference, a human would rarely do a diagonal, quickly pressing the left and right keys while his/her attention is analyzing remote areas of the screen. A person can perform a diagonal with equal efficiency than a robot, but at the cost of concentrating all attention on the narrowest area, maintaining a precise coordination of turns and trajectory.

3.8.7 Maze Navigation

There is no problem solving in nature, just adaptation. A frog, evolutionary biologists may argue, is not "a solution to the problem of being a frog", and it is not appropriate to say for example that it has solved the problem of jumping, or "being green". In the same vein, none of the behaviors we have observed among Tron agents have evolved deliberately, but rather as a consequence of the fight for survival.

Tron agents perform known basic robotics behaviors, such as obstacle avoidance or wall following, not because they have solved those problems, but rather as a consequence of evolving for survival.

Figure 3.42 shows a Tron agent that has been inserted on a maze. This agent visits some of the various passages, backtracks, then finally finds the exit and leaves. Approximately

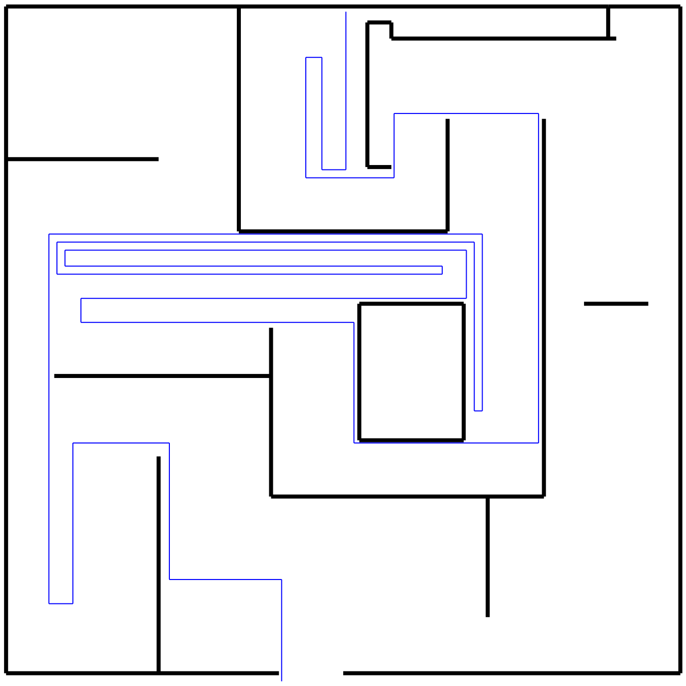


Figure 3.42: A Tron agent navigates out of a maze. It uses its own traced movements to explore until it finds the exit. There was never a purpose or explicit fitness reward for navigating mazes; instead, this agent behaves in this way as a result of its adaptation to the environment of Tron. The agent leaves the maze through the exit only because it has evolved to avoid confined spaces.

one every 10 evolved agents is able to find the exit to this particular maze. But the reason they do so is a need for survival, for there was never any payoff to do it.

The fact that the agent actually leaves the maze through the exit is particularly interesting. Why leave? Humans observing this behavior might interpret that there is a will to escape, a need for freedom; but on a deeper sense these are just consequences, *as all animal behaviors are,* of their being adapted to a competitive environment.

3.9 Discussion

3.9.1 Adapting to the Real Problem

We use a state-of-the art evolutionary algorithm, continuously being trained by self-play, as our novelty engine; but the players generated have a broad RS distribution: some of them are very good against humans and others are very bad. In all, even with feedback from the human results (by using champions as trainers, or later, by allowing champions to reproduce) the average RS of the robots produced is below the performance of Tron as a whole (section 3.7).

A Tron playing system relying solely on self-play would not be able to improve beyond this rate. *It was the power of selection coming from interaction with a live human population that drove the system up to an expert level.*

3.9.2 Evolution as Mixture of Experts

As perceived by an individual user, our virtual Tron plays differently every time, as agents are switched before each game. Simultaneously, due to the evolutionary process, the overall level of play increases. This heterogeneous behavior is part of the success; the strength of an individual strategy is valid only *given the collective behavior of all other agents* — if the same strategy were used over and over, humans would quickly adapt and the artificial opponent would appear boring and repetitive. This amounts to a *mixture of experts* architecture [69], but here the mixture is a consequence of evolution, of agents exploiting different niches created by their opponents. No single strategy can dominate, as long as there are humans who learn to respond to it, bringing down its fitness, making the way for new ones to take its place.

An interesting question for further study is to look for those emergent niches: is it possible that some agents may have adapted to specific subsets of humans? It is conceivable for example, that some agents are better against people with certain styles — novices for example, or aggressive players.

3.9.3 Human-Machine Coevolution

The word *coevolution* is used in computer science literature to describe relative fitness situations, where an individual's value depends upon the population itself. But the present work is the first example, to the best of our knowledge, of coevolution between an agent and an animal species. Agents are selected, through the fitness rules coded into the system, based only on their interaction with humans.

With Tron we are proposing a new paradigm for evolutionary computation: creating niches where agents and humans interact, leading to the evolution of the agent species. There are two main difficulties introduced when one attempts this type of coevolution against real people:

- Interactions with humans are a sparse resource.

- Opponents are random and known tournament techniques for coevolution become infeasible.

The first problem is common to all applications that wish to learn from a real, or even simulated, environment: interactions are slow and costly. We address this problem by nesting an extra loop of coevolution: while the system is waiting for human opponents, it runs many generations of agent-agent coevolution.

The second problem led us to develop a new evaluation strategy, based on the paired comparisons statistics. With it we were able to successfully select the best strategies, pushing the system to the level of a top 3% human.

The differences between self evaluation and human evaluation, studied in section 3.7, indicate, on the one hand, that evolving Tron agents by playing each other was not sufficient, as the top agents are usually not so special against people. But on the other, some of them are good, so expertise against other robots and expertise against people are not independent variables either.

We think that this is the general case: evolutionary computation is useful in domains that are not entirely unlearnable; at the same time, there is no substitute for the real experience: simulation can never be perfect.

We have also been able to show here, how most humans — at least those who stay for a while — learn from their interaction with the system; some of them quite significantly. Even though the system was not designed as a training environment for people, but rather simply as an artificial opponent, the implications for human education are exciting: evolutionary techniques provide us with a tool for building adaptive environments, capable of challenging humans with increased efficiency by interacting with a large group of people.

3.9.4 Human Motivations

The dynamics of the human population are complicated. New players arrive at a steady pace, but at the same time many veterans keep coming back (fig. 3.15), resulting in a relatively stable experience distribution. The median is 87 games, meaning that half of the games are played by users with a previous experience of 87 or more matches (25% with 387 or more).

The uniquely adaptive quality of the Tron web site is one of the reasons for such enthusiasm. One of our visitors described it in an eloquent e-mail message:

> "I can actually see that the best robots now are better than the best robots of yesterday. (Specifically, all of a sudden when I logged in at 7PM yesterday, I could swear that there were sentient beings behind some of the robots)"[8] .

[8] M. Jacobs, 1998.

Chapter 4

Conclusions

4.1 Discovery in AI

Intelligence defined as symbolic reasoning was the foundation of the first works in AI: playing chess, solving logic problems, following formal rules, were thought to epitomize intelligent behavior. Reasoning capacity was traditionally considered to be the difference between intelligent Man and unintelligent animal.

Doug Lenat claimed that his *Automated Mathematician* (AM) program [91], seeded with 115 elementary set theory concepts, was able to "discover" natural numbers and formulate interesting concepts in basic number theory such as addition, multiplication, etc.; even prime numbers. Lenat thought that the fundamental rule of intelligence, namely *heuristic search*, had been found. The follow-up project, EURISKO, would discover a domain's heuristics by itself, thus enabling discovery in any field. To discover mathematics for example, one should be able to run EURISKO with the initial axioms, and the program would eventually find both the subject's heuristics and facts, each one reinforcing the other. Lenat's hopes failed, though, and EURISKO did not live up to its expectations [90]. From this and other failures, AI shifted focus, over the ensuing decades, towards programmed expertise rather than discovery.

By showing how evolutionary entities find and exploit emergent properties of reality, we go back to Lenat's idea of AI as discovery. From the human perspective, complex is what is not obvious, what is entangled and difficult; emergent properties are the complex ones, those that need intelligence to be discovered and assimilated. With a growing understanding of what complexity and emergence are, and how they are generated, Artificial Life methods bring a new perspective on discovery as adaptation.

The goal of AI should not be to program situated agents, but to make them adaptive. Manually coding the program with all the information an organism needs becomes infeasible because it is too much — and because of the Red Queen problem: by the time we finish, the environment has changed. The code for a complete real agent, a horse for example, may well be impossible to write. Even if we knew how to, a team of programmers could not debug so many interdependent lines of code [111]. But an adaptive agent, as natural evolution shows, is capable of doing so.

4.2 From Discovery to Abstraction

ALife is founded upon a different paradigm than classic AI: it considers human intelligence as a small part logic, and a large part interaction with the habitat — embodiment. Evolutionary methods have shown the potential to develop artificial entities adapted to complex habitats and discover their rules. This is, however, only part of the problem, because it is not known how to abstract those found rules and incorporate them into the agent as modules, as new words in the language.

Here we have shown that recombination plays a fundamental role, acting as a simple way of finding and reusing higher-level descriptions and subsolutions. One of the reasons why agents can find and exploit emergent rules on their domains is that there is a partial reduction of complexity coming from the crossover operator, which replicates useful components at random. The question for the future is: how can these emergent rules be assimilated by the agent? ALife has only started to look at this problem, sometimes referred to as the *modularity* problem [5, 74].

The bulk of evidence coming from ALife work, including the one described here, supports the g-t-r hypothesis of Universal Darwinism: it is plausible that selection and recombination, in the scope of a challenging environment, can lead to increasing levels of complexity. But it also shows that this is still a crude model of evolution. Two big questions remain:

- How are discoveries assimilated into higher level descriptions?

  Take the genome of mammals as an example. It is organized to protect a core part of the genotype, while encouraging mutation on regulation factors. The result is a basic template that allows the exploration of morphology without changing the basic components [17]. Horizontal gene transfer, which leads to inter-species recombinations, has been controlled by sexual reproduction, to make it happen only within the species. Thus not only emergent complex components have been found, such as organs, but they have been incorporated into the representation, and the language of mutation and recombination itself has been reformulated.

- How can evolution be described and studied at the level of whole systems instead of agent-environment interaction?

> Coevolution (as a subfield of evolutionary computation) has obtained interesting results that could be pointing in the right general direction. These experiments do not have a drastic agent/environment distinction. Instead, the environment is the result of the collective behavior of the population. This configuration, when successful, creates a continuous challenge at the right level, leading to agents that climb the ladder of complexity. Coevolution thus breaks the first mold, that of a changing agent in a rigid environment. But real evolution involves simultaneous co-adaptation throughout all levels of organization, from the individual gene to the symbiotic association of species.

Expanded views of evolution, both from biology [92, 97, 99] and ALife [138] that look at collective evaluations, symbiosis and cooperations effects, and "evolution of evolvability" [17, 31] point towards the more general understanding of natural and artificial evolution that is necessary to address these problems.

4.3 The Humans in the Loop

With the reciprocal learning environments presented in this thesis, we have described new ways to exploit the enormous potential of feedback loops between human intelligence and evolutionary methods. The "blind watchmaker" algorithms [30, 86], and TD-Leaf's induction of the relative values of chess figures from games against humans [7] were previous indications of this potential.

EvoCAD (section 2.9) shows how a design tool can employ AI to generate creative solutions for engineering problems. Researchers in design usually reject the concept of artificial creativity, yet the diverse characteristics of computer and human problem-solving lead to solutions that are radically different. If our experiments can be described as generating surprise and innovation [116] then the goal of artificial creativity might not be impossible, after all.

In EvoCAD, human and machine take turns in trying to bring a design closer to a final goal. But Tron puts human and machine in a collaboration of a larger scale.

The cooperative effect of large groups of humans working together on the Internet is known, but Tron is the first example of an adaptive system that harvests voluntary contributions of intelligence from its users. The technique also induced learning in humans, suggesting that the coevolutionary dynamics can produce new kinds of educational and entertainment environments through coadaptation between machines and humans. Sklar and Pollack [126] have begun an effort to create educational environments for people based on the type of mutually adaptive challenge first demonstrated by Tron.

The Internet is a new kind of environment where people and software interact in an unprecedented scale. Thus the potential for a new kind of intelligent software that — as exemplified by our Tron domain — thrives on the niches of a virtual ecology that mixes natural and artificial life forms.

4.4 The Reality Effect

We have described in detail some of the solutions encountered by our test domains: the appearance of nested levels of complexity in Lego structures, the reuse and adaptation of components, the emergence of basic navigation behaviors such as wall-following, spiraling, maze navigation and space filling, amongst Tron agents.

The Brooksian idea that an agent's complexity sometimes comes from the environment more than the agent itself is supported by the evidence of Tron robots: these stateless agents have no internal means for storing plans or previous decisions, yet were shown (section 3.8) to reliably evolve behaviors involving choreographed sequences of moves.

By adapting to the real problem, rather than a metaphor or an incomplete simulation, the applied aspect of evolutionary methods comes to life, which could lead to a variety of applications.

Fully Automated Design, the idea of computers designing complete, functional artifacts, based on constraints and goals defined externally, was shown here with Lego structures. Lipson and Pollack extended this idea by showing how a static motion simulator can be used to coevolve morphology and brain of walking creatures [94]. Automated design is central to evolutionary robotics, for brains need bodies to inhabit, as well as evolutionary design, which needs to break the mold of pre-imposed problem decomposition, and play more freely with components and goals.

Together with work by other researchers such as Sims and Thompson our thesis deals with what we have called *the reality effect*: when evolution interacts with a large, complex environment like those typically generated by the world around us, complex solutions appear that exploit emergent properties of the domain in surprising new ways.

Bibliography

[1] 20th Century Fox (1966) *Fantastic Voyage*. R. Fleischer, director (Film).

[2] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993) *Network Flows*. Prentice Hall, Englewood Cliffs.

[3] Ali, A., Helgason, R. V., Kennington, J. L. and Lall, H. (1980) Computational comparison among three multicommodity network flow algorithms. *Operations Research* **28**: 995–1000.

[4] Angeline, P. J. and Pollack, J. B. (1993) Competitive environments evolve better solutions for complex tasks. Forrest, S. (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, Calif., 264–270.

[5] Angeline, P. J. and Pollack, J. B. (1994) Coevolving high-level representations. *Artificial life III*. Addison-Wesley, Reading, Mass., 55–71.

[6] Axelrod, R. (1987) The evolution of strategies in the iterated prisoner's dilemma. Davis, L. (ed.), *Genetic Algorithms and Simulated Annealing*, Pitman: London.

[7] Baxter, J., Tridgell, A. and L.Weaver (1998) TDLeaf(λ): Combining temporal difference learning with game-tree search. *Proceedings of the Ninth Australian Conference on Neural Networks*. 168–172.

[8] Beasley, D., Bull, D. R. and Martin, R. R. (1993) A sequential niche technique for multimodal function optimization. *Evolutionary Computation* **1**(2): 101–125.

[9] Belew, R. K., McInerney, J. and Schraudolf, N. (1990) Evolving networks, using the genetic algorithm with connectionist learning. Technical Report CSE-CS-174, UCSD.

[10] Bennet, C. H. (1985) *Emerging Syntheses in Science*. Pines.

[11] Bentley, P. (ed.) (1999) *Evolutionary Design by Computers*. Morgan-Kaufmann, San Francisco.

[12] Bentley, P. J. (1996) *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*. Ph.D. thesis, Division of Computing and Control Systems, School of Engineering, The University of Huddersfield.

[13] Bezerra, C. and Owen, C. L. (2000) Evolutionary structured planning. Gero, J. S. (ed.), *Artificial Intelligence in Design '00*. Kluwer Academic, 287–307.

[14] Brooks, R. (1991) Intelligence without reason. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*. IJCAI, San Mateo, Calif., 569–595.

[15] Brooks, R. (1991) Intelligence without representation. *Artificial Intelligence* **47**(1-3): 139–160.

[16] Brooks, R. A. (1986) A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* **2**(1): 14–23.

[17] Carroll, S. B. (2000) Endless forms: The evolution of gene regulation and morphological diversity. *Cell* **101**: 577–580.

[18] Castro, J. and Nabona, N. (1996) An implementation of linear and nonlinear multicommodity network flows. *European Journal of Operational Research* **92**: 37–53.

[19] Chaitin, G. J. (1966) On the length of programs for computing finite binary sequences. *Journal of the ACM* **13**(4): 547–569.

[20] Chapman, C. D., Saitou, K. and Jakiela, M. J. (1993) Genetic algorithms as an approach to configuration and topology design. Gilmore, B. (ed.), *Advances in Design Automation*. American Society of Mechanical Engineers (ASME), New York, no. 65:1 in series DE, 485–498.

[21] Cherkassky, B. V. and Goldberg, A. V. (1997) On implementing push-relabel method for the maximum flow problem. *Algorithmica* **19**: 390–410.

[22] Cliff, D., Harvey, I. and Husbands, P. (1993) Explorations in evolutionary robotics. *Adaptive Behavior* **2**(1): 71–108.

[23] Cliff, D., Husbands, P. and Harvey, I. (1992) Analysis of evolved sensory-motor controllers. Technical Report Technical Report CSRP 264, University of Sussex School of Cognitive and Computing Sciences.

[24] Cliff, D. and Miller, G. (1995) Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. Morán, F., Moreno, A., Merelo, J. J. and Chacón, P. (eds.), *Advances in Artificial Life: Third European Conference on Artificial Life*. Springer, Berlin, New York, no. 929 in Lecture Notes in Computer Science, 200–218.

[25] Cliff, D. and Miller, G. (1996) Co-evolution of pursuit and evasion II: Simulation methods and results. *From Animals to Animats 4*. MIT Press, 506–515.

[26] Cliff, D. and Noble, J. (1997) Knowledge-based vision and simple visual machines. *Philosophical Transactions of the Royal Society of London: Series B* **352**: 1165–1175.

[27] Cormen, T. H., Leiserson, C. and Rivest, R. L. (1989) *Introduction to Algorithms*. MIT Press - McGraw Hill.

[28] Cziko, G. (1995) *Without Miracles: Universal Selection Theory and the Second Darwinian Revolution*. MIT Press, Cambridge, USA.

[29] Dawkins, R. (1983) Universal darwinism. Bendall, D. S. (ed.), *Evolution from Molecules to Man*, Cambridge University Press, Cambridge. 403–425.

[30] Dawkins, R. (1987) *The Blind Watchmaker*. W. W. Norton, New York.

[31] Dawkins, R. (1996) *Climbing Mount Improbable*. W. W. Norton, New York.

[32] Di Paolo, E. A. (2000) Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. Meyer, J., Berthoz, A., Floreano, D., Roitblat, H. and Wilson, S. W. (eds.), *From Animals to Animats 6*. MIT Press, Cambridge (Mass), London (England), 440–449.

[33] Edmonds, B. (1999) *Syntactic Measures of Complexity*. Ph.D. thesis, University of Manchester, Department of Philosophy.

[34] Elo, A. E. (1986) *The Rating of Chessplayers, Past and Present*. Arco Pub., New York, 2nd ed.

[35] Fahlman, S. E. (1974) A planning system for robot construction tasks. *Artificial Intelligence* **5**: 1–49.

[36] Floreano, D. (1998) Evolutionary robotics in artificial life and behavior engineering. Gomi, T. (ed.), *Evolutionary Robotics*, AAI Books, Ontario (Canada).

[37] Floreano, D. and Mondada, F. (1994) Automatic creation of an autonomous agent: Genetic evolution of a neural network driven robot. D. Cliff, P. H., Meyer, J. and Wilson, S. W. (eds.), *From Animals to Animats 3*. MIT Press, Bradford Books.

[38] Floreano, D. and Mondada, F. (1996) Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics* .

[39] Floreano, D., Nolfi, S. and Mondada, F. (1998) Competitive co-evolutionary robotics: From theory to practice. *From Animals to Animats 4*. MIT Press.

[40] Fogel, D. B. (2000) Evolving a checkers player without relying on human expertise. *Intelligence* **11**(2): 20–27.

[41] Forbus, K. (1984) Qualitative process theory. *Artificial Intelligence* **24**: 85–168.

[42] Fukuda, T. and Kawauchi, Y. (1990) Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*. 662–667.

[43] Funes, P. (1996) The Tron game: an experiment in artificial life and evolutionary techniques. (Unpublished).

[44] Funes, P. (2000) Measuring progress in coevolutionary competition. Meyer, J., Berthoz, A., Floreano, D., Roitblat, H. and Wilson, S. W. (eds.), *From Animals to Animats 6*. MIT Press, Cambridge (Mass), London (England), 450–459.

[45] Funes, P., Lapat, L. B. and Pollack, J. B. (2000) EvoCAD: Evolution-assisted design. *Artificial Intelligence in Design'00 (Poster Abstracts)*. Key Centre of Design Computing and Cognition, University of Sidney, 21–24.

[46] Funes, P. and Pollack, J. B. (1997) Computer evolution of buildable objects. Husbands, P. and Harvey, I. (eds.), *Fourth European Conference on Artificial Life*. MIT Press, Cambridge, 358–367.

[47] Funes, P. and Pollack, J. B. (1998) Componential structural simulator. Technical Report CS-98-198, Brandeis University Department of Computer Science.

[48] Funes, P. and Pollack, J. B. (1998) Evolutionary body building: Adaptive physical designs for robots. *Artificial Life* **4**(4): 337–357.

[49] Funes, P. and Pollack, J. B. (1999) Computer evolution of buildable objects. Bentley, P. (ed.), *Evolutionary Design by Computers*, Morgan-Kaufmann, San Francisco. 387 – 403.

[50] Funes, P., Sklar, E., Juillé, H. and Pollack, J. B. (1998) Animal-animat coevolution: Using the animal population as fitness function. *From Animals to Animats 5*. MIT Press, Cambridge, MA, University of Zurich, 525–533.

[51] Funes, P., Sklar, E., Juillé, H. and Pollack, J. B. (1998) Animal-animat coevolution: Using the animal population as fitness function. Pfeiffer, Blumberg, Wilson and Meyer (eds.), *From Animals to Animats 5*. MIT Press.

[52] Gardin, F. and Meltzer, B. (1989) Analogical representations of naive physics. *Artificial Intelligence* **38**: 139–159.

[53] Gell-Mann, M. and Lloyd, S. (1996) Information measures, effective complexity, and total information. *Complexity* **2**(1): 44–52.

[54] Glickman, M. E. (1999) Parameter estimation in large dynamic paired comparison experiments. *Applied Statistics* **48**: 377–394.

[55] Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading.

[56] Gould, S. and Vrba, E. (1982) Exaptation - a missing term in the science of form. *Paleobiology* **8**: 4–15.

[57] Gould, S. J. (1996) *Full house: the spread of excellence from Plato to Darwin.*. Harmony Books, New York.

[58] Grassberger, P. (1990) Information and complexity measures in dynamical systems. *Information Dynamics*. Plenum Press, New York.

[59] Grigoriadis, M. D. and Khachiyan, L. G. (1995) An exponential-function reduction method for block-angular convex programs. *Networks* **26**: 59–68.

[60] Gruau, F. (1992) Cellular encoding of genetic neural networks. Technical Report 92.21, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon.

[61] Gruau, F. (1993) Genetic synthesis of modular neural networks. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan-Kaufman, 318–325.

[62] Harvey, I. (1997) Artificial evolution for real problems. Gomi, T. (ed.), *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97)*. AAI Books, Kanata, Ontario Canada, 127–149.

[63] Heylighen, F. (1989) Self-organization, emergence and the architecture of complexity. *Proceedings of the First European Conference on System Science*. AFCET, Paris, 23–32.

[64] Heylighen, F. (1999) The growth of structural and functional complexity during evolution. Heylighen, F., Bollen, J. and Riegler, A. (eds.), *The Evolution of Complexity*, Kluwer Academic, Dordrecht-Boston-London. 17–43.

[65] Hillis, D. (1991) Co-evolving parasites improves simulated evolution as an optimization procedure. C. Langton, C. Taylor, J. F. and Rasmussen, S. (eds.), *Artificial Life II*, Addison-Wesley, Reading, MA.

[66] Hornby, G. S., Lipson, H. and Pollack, J. B. (2001) Evolution of generative design systems for modular physical robots. *IEEE International Conference on Robotics and Automation*. (to appear).

[67] Husbands, P. and Harvey, I. (1992) Evolution versus design: Controlling autonomous robots. *Proceedings of the Third Annual Conference on Artificial Intelligence, Simulation and Planning*. IEEE Press, 139–146.

[68] Iusem, A. and Zenios, S. (1995) Interval underrelaxed Bregman's method with an application. *Optimization* **35**(3): 227.

[69] Jacobs, R. A., Jordan, M. I., Nowlan, S. J. and Hinton, G. E. (1991) Adaptive mixtures of local experts. *Neural Computation* **3**: 79–87.

[70] Jakobi, N. (1994) Evolving sensorimotor control architectures in simulation for a real robot. Master's thesis.

[71] Jakobi, N. (1997) Half-baked, ad hoc, and noisy: minimal simulations for evolutionary robotics. Husbands, P. and Harvey, I. (eds.), *Fourth European Conference on Artificial Life*. MIT Press, 348–357.

[72] Jakobi, N., Husbands, P. and Harvey, I. (1995) Noise and the reality gap: The use of simulation in evolutionary robotics. Morán, F., Moreno, A., Merelo, J. J. and Chacón, P. (eds.), *Advances in Artificial Life: Third European Conference on Artificial Life*. Springer, Berlin, New York, no. 929 in Lecture Notes in Computer Science, 704–720.

[73] Joe, H. (1990) Extended use of paired comparison models, with application to chess rankings. *Applied Statistics* **39**(1): 85–93.

[74] Juillé, H. and Pollack, J. B. (1996) Co-evolving intertwined spirals. *Proceedings of the Sixth International Conference on Genetic Algorithms*. 351–358.

[75] Juillé, H. and Pollack, J. B. (1996) Dynamics of co-evolutionary learning. *From Animals to Animats 4*. MIT Press, 526–534.

[76] Kawauchi, Y., Inaba, M. and Fukuda, T. (1999) Genetic evolution and self-organization of cellular robotic system. *JSME Int. J. Series C. (Dynamics, Control, Robotics, Design & Manufacturing)* **38**(3): 501–509.

[77] Keirsey, D. M. (1999) Involution: On the structure and process of existence. Heylighen, F., Bollen, J. and Riegler, A. (eds.), *The Evolution of Complexity*, Kluwer Academic, Dordrecht-Boston-London. 45–57.

[78] Kitano, H. (1990) Designing neural network using genetic algorithm with graph generation system. *Complex Systems* **4**: 461–476.

[79] Kolmogorov, A. N. (1965) Three approaches to the quantitative definition of information. *Problems of Information Transmission* **1**(1): 1–11.

[80] Kolmogorov, A. N. (1983) Combinatorial basis of information theory and probability theory. *Russian Mathematical Surveys* **38**: 29–40.

[81] Komosinski, M. and Ulatowski, S. (1999) Framsticks: towards a simulation of a nature-like world, creatures and evolution. Floreano, D., Nicoud, J.-D. and Mondada, F. (eds.), *Advances in Artificial Life: 5th European Conference on Artificial Life*. Springer-Verlag, vol. 1674 of *Lecture Notes in Artificial Intelligence*, 261–265.

[82] Koppel, M. (1987) Complexity, depth, and sophistication. *Complex Systems* **1**: 1087–1091.

[83] Kotay, K., Rus, D., Vona, M. and McGray, C. (1998) The self-reconfiguring robotic molecule. *1998 IEEE International Conference on Robotics and Automation,*. Robotics and Automation Society.

[84] Koza, J. (1992) *Genetic Programming*. MIT Press, Cambridge.

[85] Koza, J. R. (1990) Evolution of subsumption using genetic programming. Varela, F. J. and Bourgine, P. (eds.), *Toward a practice of autonomous systems: First European Conference on Artificial Life*. MIT Press, 110–119.

[86] K.Sims (1991) Artificial evolution for computer graphics. *Computer Graphics (Siggraph '91 proceedings)*. 319–328.

[87] Langton, C. (1989) Artificial life. Langton, C. (ed.), *Artificial Life: the proceedings of an Interdisciplinary Workshop*. Addison-Wesley, 1–47.

[88] Lee, W., Hallam, J. and Lund, H. (1996) A hybrid GP/GA approach for co-evolving controllers and robot bodies to achieve fitness-specified tasks. *Proceedings of IEEE 3rd International Conference on Evolutionary Computation*. IEEE Press, Piscataway, N.J., 384–389.

[89] Leighton, T., Makedon, F., Plotkin, S., Stein, C., Tardos, E. and Tragoudas, S. (1995) Fast approximation algorithms for muticommodity flow problems. *Journal of Computer and Systems Sciences* **50**: 228–243.

[90] Lenat, D. and Brown, J. (1984) Why AM and EURISKO appear to work. *Artificial Intelligence* **23**: 269–294.

[91] Lenat, D. B. (1977) The ubiquity of discovery. *International Joint Conference on Artificial Intelligence (5th, 1977)*. IJCAI, 1093–1105.

[92] Lewontin, R. C. (2000) *The triple helix : gene, organism, and environment*. Harvard University Press, Cambridge, Mass.

[93] Lieberman, H. (1997) Autonomous interface agents. *ACM Conference on Human-Computer Interface*.

[94] Lipson, H. and Pollack, J. B. (2000) Automatic design and manufacture of robotic lifeforms. *Nature* **406**(6799): 974–978.

[95] Lund, H. (1995) Evolving robot control systems. Alander, J. T. (ed.), *Proceedings of the First Nordic Workshop on Genetic Algorithms and their Applications*. University of Vaasa, Vaasa.

[96] Lund, H., Hallam, J. and Lee, W. (1997) Evolving robot morphology. *Proceedings of IEEE Fourth International Conference on Evolution*. IEEE Press.

[97] Margulis, L. (1993) *Symbiosis in cell evolution: microbial communities in the Archean and Proterozoic eons*. Freeman, New York, 2nd ed.

[98] Mataric, M. J. and Cliff, D. (1996) Challenges in evolving controllers for physical robots. *Robotics and Autonomous Systems* **19**(1): 67–83.

[99] Maynard Smith, J. and Szathmáry, E. (1997) *The major transitions in evolution*. Oxford University Press, Oxford.

[100] McCabe, T. J. (1976) A complexity measure. *IEEE Transacions on Software Engineering* **2**(4): 308–320.

[101] Miller, G. F. and Cliff, D. (1994) Protean behavior in dynamic games. Cliff, D., Husbands, P., Meyer, J. and Wilson, S. (eds.), *From Animals to Animats 3*, MIT Press.

[102] Minsky, M. (1986) *The Society of Mind*. Simon & Schuster, New York.

[103] Newborn, M. (1996) *Kasparov vs. Deep Blue: Computer Chess Comes of Age*. Springer Verlag, New York.

[104] Pamecha, A., Chiang, C., Stein, D. and Chirikjian, G. S. (1996) Design and implementation of metamorphic robots. *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*.

[105] Park, H. S. and Gero, J. S. (2000) Categorisation of shapes using shape features. Gero, J. S. (ed.), *Artificial Intelligence in Design '00*. Kluwer Academic, 203–223.

[106] Plotkin, H. C. (1993) *Darwin Machines and the Nature of Knowledge*. Harvard University Press.

[107] Pollack, J. B. and Blair, A. (1997) Why did TD-Gammon work? *Advances in Neural Information Processing Systems* **9**: 10–16.

[108] Pollack, J. B., Blair, A. and Land, M. (1996) Coevolution of a backgammon player. Langton, C. (ed.), *Artificial Life V*. MIT Press.

[109] Pollack, J. B. and Blair, A. D. (1998) Coevolution in the successful learning of backgammon strategy. *Machine Learning* **32**: 225–240.

[110] Pollack, J. B., Lipson, H., Ficici, S., Funes, P., Hornby, G. and Watson, R. (2000) Evolutionary techniques in physical robotics. Miller, J. (ed.), *Evolvable Systems: from biology to hardware*. Springer-Verlag, no. 1801 in Lecture Notes in Computer Science, 175–186.

[111] Pollack, J. B., Lipson, H., Funes, P., Ficici, S. G. and Hornby, G. (1999) Coevolutionary robotics. Koza, J. R., Stoica, A., Keymeulen, D. and Lohn, J. (eds.), *The First NASA/DoD Workshop on Evolvable Hardware*. IEEE Press.

[112] Ray, T. (1992) An approach to the synthesis of life. C. Langton, C. Taylor, J. F. and Rasmussen, S. (eds.), *Artificial Life II*, Addison-Wesley, Reading, MA.

[113] Ray, T. S. (1994) An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life* **1**: 179–209.

[114] Reynolds, C. (1994) Competition, coevolution, and the game of tag. *Artificial Life IV*, MIT Press. 59–69.

[115] Rich, E. and Kight, K. (1991) *Artificial Intelligence*. McGraw-Hill, New York, 2nd ed.

[116] Ronald, E. M. A. and Sipper, M. (2000) Engineering, emergent engineering, and artificial life: Unsurprise, unsurprising surprise, and surprising surprise. , M. A. B., McCaskill, J. S., Packard, N. H. and Rasmussen, S. (eds.), *Artificial Life VII*. MIT Press, Cambridge, 523–528.

[117] Rosca, J. (1996) Generality versus size in genetic programming. *Proceedings of the Genetic Programming 1996 Conference*. MIT Press.

[118] Rosin, C. D. (1997) *Coevolutionary Search Among Adversaries*. Ph.D. thesis, University of California, San Diego.

[119] Rosin, C. D. and Belew, R. K. (1995) Methods for competitive co-evolution: finding opponents worth beating. *Proceedings of the 6th International Conference on Genetic Algorithms*. Morgan Kaufman, 373–380.

[120] Schoenauer, M. (1996) Shape representations and evolution schemes. Fogel, L. J., Angeline, P. J. and Back, T. (eds.), *Proceedings of the 5th Annual Conference on Evolutionary Programming*. MIT Press.

[121] Shannon, C. E. (1948) A mathematical theory of communication. *Bell System Technical Journal* **27**: 379–423,623–656.

[122] Simon, H. A. (1962) The architechture of complexity. *Proceedings of the American Philosophical Society* **106**: 467–482.

[123] Sims, K. (1994) Evolving 3D morphology and behavior by competition. Brooks, R. and Maes, P. (eds.), *Artificial Life IV*. MIT Press, 28–39.

[124] Sims, K. (1994) Evolving virtual creatures. *Computer Graphics*, Annual Conference Series.

[125] Sklar, E., Blair, A. D., Funes, P. and Pollack, J. B. (1999) Training intelligent agents using human internet data. Liu, J. and Zhong, N. (eds.), *Intelligent agent technology: systems, methodologies, and tools*. World Scientific, Singapore, River Edge, NJ.

[126] Sklar, E. and Pollack, J. B. (2000) A framework for enabling an internet learning community. *Educational Technology and Society* **3**(3): 393–408.

[127] Solomonoff, R. J. (1964) A formal theory of inductive inference I, II. *Information Control* **7**: 1–22, 224–254.

[128] Sutton, R. (1988) Learning to predict by the methods of temporal differences. *Machine Learning* **3**: 9–44.

[129] Temperley, H. N. (1981) *Graph Theory and Applications*. Ellis Horwood, Chichester.

[130] Tesauro, G. (1990) Neurogammon wins computer olympiad. *Neural Computation* **1**: 321–323.

[131] Tesauro, G. (1992) Practical issues in temporal difference learning. *Machine Learning* **8**: 257–277.

[132] Tesauro, G. (1995) Temporal difference learning and TD-Gammon. *Communications of the ACM* **38**(3): 58–68.

[133] Thompson, A. (1995) Evolving electronic robot controllers that exploit hardware resources. Morán, F., Moreno, A., Merelo, J. J. and Chacón, P. (eds.), *Advances in Artificial Life: Third European Conference on Artificial Life*. Springer, Berlin, New York, no. 929 in Lecture Notes in Computer Science, 640–656.

[134] Thompson, A. (1998) *Hardware evolution : Automatic design of electronic circuits in reconfigurable hardware by Artificial Evolution*. Springer, London; New York.

[135] Toulmin, S. (1953) *The Philosophy of Science*. Hutchinson, London.

[136] V'yugin, V. V. (1999) Algorithmic complexity and stochastic properties of finite binary sequences. *Computer Journal* **42**(4): 294–317.

[137] Walt Disney Studios (1982) *Tron*. S. Lisberger, director (Film).

[138] Watson, R. A. and Pollack, J. B. (2000) Symbiotic combination as an alternative to sexual recombination in genetic algorithms. Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J. J. and Schwefel, H.-P. (eds.), *Proceedings of Parallel Problem Solving from Nature VI*. Springer Verlag, no. 1917 in Lecture Notes in Computer Science.

[139] Yim, M. (1995) *Locomotion With A Unit-Modular Reconfigurable Robot*. Ph.D. thesis, Stanford University, Department of Computer Science.

[140] Yim, M., Duff, D. G. and Roufas, K. D. (2000) PolyBot: a modular reconfigurable robot. *Proceedings : 2000 IEEE International Conference on Robotics and Automation*. Robotics and Automation Society, Piscataway, NJ.

[141] Zienkiewicz, O. (1977) *The Finite Element Method in Engineering Science*. McGraw-Hill, New York, 3rd ed.