Building processes optimization : Toward an artificial ontogeny based approach

Alexandre Devert

Directed by Marc Schoenauer and Nicolas Bredèche

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

June 21, 2009

Ecole Doctorale d'Informatique
Université Paris-Sud 11

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche SACLAY – ÎLE-DE-FRANCE

Contents

1 Introduction to Evolutionary Design **5**

1.1 An evolutionary algorithms primer 7

1.1.1 Description 8

1.1.2 Lamarckian and Baldwinian evolutions 9

1.1.3 On the design of an evolutionary algorithm 10

1.2 Simulation and evaluation 11

1.2.1 Design of an evaluation function 13

1.2.2 The usage of simulation 14

1.3 Representations for Evolutionary Design 15

1.3.1 Block stacks 15

1.3.2 Explicit and implicit representations 18

1.3.3 Modularity 19

1.3.4 Tree structures 21

1.3.5 Grammars and rewriting systems 23

1.3.6 The evolvability issue 26

1.3.7 Cellular representations 26

1.4 Preview 29

2 A guided tour of Evolutionary Design **31**

2.1 Explicit representations 31

2.1.1 Block stacks 31

2.1.2 Lego structures 32

2.1.3 Antennae 33

2.1.4 Shapes 34

2.1.5 Robots 35

2.1.6 Tensegrity structures 36

2.1.7 Topological Optimum Design 37

2.1.8 Optical fibers design 38

2.1.9 Blade shape design for wind turbines 39

2.2 Implicit representations 40

2.2.1 Karl Sims' creatures 40

2.2.2 Hornby's L-systems 41

2.2.3 Environment aware L-systems 44

2.3 Cellular representations 45

2.3.1 Virtual creatures 46

2.3.2 Skyscraper frames 47

2.3.3 Truss structures 48

2.3.4 Environment awareness 50

2.4 Conclusion 51

3 An Evolutionary algorithm for blueprints **55**
3.1 The BlindBuilder representation . . . 55
3.1.1 Variation operators . . . 57
3.2 Design of simple 3d objects . . . 57
3.2.1 Experimental setup and hierarchical fitness . . . 58
3.2.2 The Pillar experiment . . . 59
3.2.3 The Bridge experiment . . . 60
3.3 An empirical analysis . . . 62
3.3.1 Experimental settings . . . 64
3.3.2 Experimental Protocol . . . 65
3.3.3 Results . . . 66
3.4 Discussion . . . 71
3.5 Conclusion . . . 72

4 Embryogenic flags **75**
4.1 Introduction to embryogenic flags . . . 76
4.2 An insight of the embryogenic flag problem . . . 78
4.2.1 Direct Solutions . . . 79
4.2.2 Non-Developmental Solutions . . . 80
4.2.3 A generic developmental solution . . . 80
4.2.4 Reaction-diffusion systems . . . 81
4.2.5 Towards non-trivial developmental solutions . . . 83
4.3 Survey of the Embryogenic Flags . . . 83
4.3.1 Update Function Models . . . 84
4.3.2 Cell Communications . . . 84
4.3.3 Neighbourhoods and Inputs . . . 85
4.3.4 Synchronous and Asynchronous Updates . . . 86
4.3.5 Initial Setup . . . 86
4.3.6 Birth, Growth and Death . . . 86
4.3.7 Stopping Criterion . . . 87
4.3.8 Self-Repairing Properties . . . 88
4.3.9 Relevance to Evolutionary Design . . . 88
4.4 Methodology . . . 90
4.4.1 The Target Patterns . . . 90
4.4.2 Stopping criterion . . . 92
4.4.3 Pattern distance . . . 93
4.4.4 Fitness function . . . 94
4.4.5 Steady state stability . . . 94
4.4.6 Parametric optimization . . . 94
4.5 Neural networks as update functions: NEAT . . . 95
4.5.1 The model . . . 95
4.5.2 Controller Optimization . . . 96
4.5.3 Experimental Setup . . . 97
4.5.4 Results and discussion . . . 98
4.6 Reservoir computing for update functions . . . 104
4.6.1 Echo State Networks . . . 104

4.6.2 Controller optimization . . . 105
4.6.3 Results and discussion . . . 106
4.7 A diffusion based approach . . . 106
4.7.1 The model . . . 108
4.7.2 Controller Optimization . . . 109
4.7.3 Results and discussion . . . 109
4.8 Development and stability . . . 115
4.8.1 Experimental setup . . . 115
4.8.2 Results and discussion . . . 115
4.9 Conclusion . . . 120

5 A cellular development model for structural truss design 125
5.1 Structural trusses . . . 125
5.1.1 Optimization of Truss Structures . . . 127
5.2 EiffelBlob . . . 127
5.2.1 Control Topologies . . . 128
5.2.2 Control and information diffusion . . . 129
5.2.3 Geometry control . . . 130
5.2.4 Mechanical feedback . . . 131
5.2.5 Initialization and halting . . . 131
5.3 Experimentations . . . 132
5.3.1 Optimization of a developement process . . . 134
5.3.2 Comparison with the Direct Representation . . . 135
5.3.3 Properties of the Developement Process . . . 138
5.4 Conclusion . . . 139

6 Discussion and Conclusion 143
6.1 Contributions . . . 143
6.1.1 Representations for Evolutionary Design . . . 143
6.1.2 Explicit Building Processes . . . 143
6.1.3 Embryogenic Flags . . . 144
6.1.4 Optimizing Cellular Development Processes . . . 145
6.1.5 Stopping Criterion . . . 146
6.1.6 A Cellular Representation for Trusses . . . 146
6.1.7 Summary . . . 147
6.2 Perspectives for cellular representations . . . 147
6.2.1 Current Issues . . . 148
6.2.2 Future Directions . . . 149

A The Springy Comb 151
A.1 Description . . . 151
A.2 Implementation . . . 153

B Blocks stacks physics 155
B.1 The model . . . 155
B.2 Force, equilibrium and balance . . . 155

C NEAT **157**
C.1 The representation 157
C.2 The variation operators 157
C.2.1 Mutations 158
C.2.2 Crossover 159
C.3 The generational operator 159
C.4 Issues 159

A long time ago, in a galaxy far, far away . . .

Star Wars Episode IV: A New Hope

1

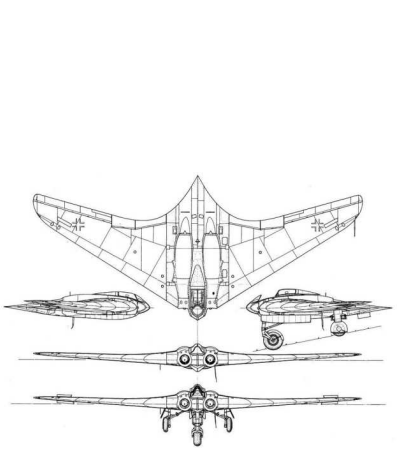
Introduction to Evolutionary Design

Most of the living beings, without predators or catastrophic events like epidemic, would see their populations growing at an exponential rate. But those beings rely on the same limited resources, which can not sustain exponentially growing populations for a long time. Thus, there is a constant struggle for the vital resources. In addition, living beings usually breed individuals very similar to themselves, but with some tiny variations. Some of those variations might be able to give a slight advantage for access to the vital resources. Then those variations are likely to spread quickly and become a definitive character of a specie. This is a very abridged introduction to the natural selection theory of Charles R. Darwin, ignoring various biological facts, like the genetic material inner functioning, the role of the environment geography, . . .

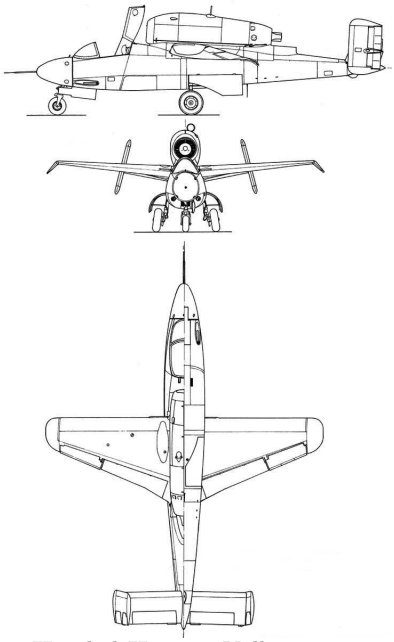
Such a summed-up version of the natural evolution is quite general, and we can replace *beings* by *machines* without much transgression to logic. In 2008, it would be yet a bit too soon to talk about self-reproducing autonomous machines like a mastered fact. But we can imagine without any difficulties a huge room filled by engineers. Each engineer conceives a plane for a predefined purpose. Those planes are *candidate solutions*. Once the conception is done, some prototypes of the corresponding planes are built. The prototypes are tested by test pilots, their designs are reviewed by an expert commission, the productions costs are estimated, etc. The candidates solutions are evaluated during an *evaluation* stage. And finally, amongst the plane designs from the engineers room, a few plane designs might be adopted and produced: this is the *selection* stage. An exceptional, versatile and cheap plane design is likely to be produced in massive quantities, while an average, specialized and expensive design will be produced in little quantities. The planes design evolved with a striking resemblance with natural evolution.

Such a caricature view, rooms filled of engineers drawing planes, actually existed during the Second World War. In Germany, there was a high pressure to compensate for the lack of resources and industrial strength by cutting edge technologies. Combined to the frantic desperation and the fury of a darned regime, this generated an explosion of planes designs. Drastic amounts of innovative, groundbreaking designs were proposed, selected an tested at that time. The invention, the evaluation and the selection stages were performed in

Figure 1.1: The *Triebflügel* concept. It was designed to takeoff vertically, on rough airfields, using low quality fuel. The three wings would rotate around the plane body, propelled by a compact ramjet at their tip. Although the Pabst ramjet was tested, this design never made its way to production.



Horten Ho 229



Heinkel He 162 *Volksjäger*

Figure 1.2: Two German aircraft of the Second World War, amongst the less loose concepts. At the end of the war, the left one was about to make its first flight, while the mass production of the one at the right side was beginning. Hundreds of even more daring designs were studied as well, but sadly, good pictures of them are quite hard to find.

very short cycles. German engineers of World War Two devised inverted arrow wings, delta wings, jet engine or rockets propelled flying wings [1], composite planes, suborbital bombers [2], asymmetric planes [3], variable geometry planes [4], or even aberrations like the *Triebflügel* design (showed on figure 1.1). An exploding, never quite equaled variety of very original designs came out from the German aircraft manufacturers. Few of those Germans designs made their way to the mass production and the battlefield, but the more reasonable ones are everywhere now: jet engine powered delta wings and inverted arrow wings, ... Indeed, most of the best Germans engineers were recruited from Germany at the end of the war, to continue their work in the Allied nations.

Thanks to the current modeling and simulation software available, together with the average computing power accessible with even a modest investment, there is much less need to build physical prototypes. Many properties of a candidate plane design are accessible by simulators, and thus it is tempting to replace the room full of engineers by some computational plane design generator and the test pilots by models and simulations. The variations of candidate plane designs would be randomly generated, and evaluated by the computers by mean of simulation. The selection would be done from a set of criteria: the flight behavior, the maximum distance, the intake capacity, etc. With blind, random variations, original solutions might be found in an automated way. This is what Evolutionary Design is all about: it is a strategy to explore in an automated way a design space that adapts toward good designs, by plugging together modeling, random variations and simulation.

In this chapter, we first introduce Evolutionary Algorithms, which are the general framework of those optimisation strategies based on random variations, evaluation and selection. Evolutionary Design is the specialization of Evolutionary Algorithms that discovers and optimizes new designs of physical objects, evaluated through simulations. After a quick introduction to numerical simulations, we discuss the outcomes of their coupling with optimization algorithms. What we believe to be the strongest issue is the *representation problem*. We propose various solutions to that problem, discussing their potential strengths and weaknesses. We conclude with an outline of the rest of the thesis.

1.1 An evolutionary algorithms primer

Since Evolutionary Design is a specialization of Evolutionary Algorithms, we provide here a description of those algorithms. Evolutionary Algorithms are more a set of algorithmic principles and mechanisms for adaptation and optimization algorithms, rather than actual algorithms: they are instances of the same *meta-heuristic*. As a consequence, one of their strengths is they can explore and optimize any kind of representation space: space of trees, space of

[1] like the Messerschmitt Me 163 *Komet* which had have been used in operation
[2] Sangër Amerika Bomber
[3] Blohm & Voss BV 141
[4] Messerschmitt Me P.1101

grammars, space of graphs, space of finite state machines, etc... This space is called the *genotypic space*, whose elements are *genotypes*. Candidates solutions are evaluated by the *evaluation function* that returns a value, the *fitness*. The fitness is an indicator of a solution quality. The evaluation function does not take necessarily the genotype as input, but a transformed version of it: the *phenotype*. For instance, a phenotype can be a shape and the genotype some control points of that shape. An Evolutionary Algorithm will try to optimize (maximize or minimize) the fitness value by searching in the genotypic space. In theory, no constraints are put on the evaluation function and the nature of the genotypic space. But after a description of the Evolutionary Algorithms, we will show that in practice, some properties are desirable in order to have a competent algorithm.

1.1.1 Description

Evolutionary Algorithms work in an iterative fashion, by maintaining a pool of candidate solutions, called a *population*. The candidate solutions are called *individuals*. Being an iterative algorithm, an initial population is needed to start the algorithm. This is the role of the *initialization operator*: it builds randomly this initial population. Then, a single population update, called a *generation*, is carried as follows:

1. Select randomly a set of individuals from the population with a bias towards the fittest individuals, with the *selection operator*.
2. Build a new set of individuals from the previously selected ones, with a *breeding operator*.
3. The new set of individuals is inserted in the population, and others individuals are removed from the population. This is the role of the *replacement operator*. The replacement operator also have a bias toward the best individuals: fittest individuals are more likely to be kept in the population.

The iterations halt when a stopping criterion is met. It could be a user provided criterion, like a given level fitness to beat. Some algorithms have an endogenous criterion: they stops iterating when no real progress in the solutions quality is likely to occur.

The breeding operator is responsible of the variation, building new candidates solutions from a pool of previous candidates solutions. The *mutation operator* takes one individual and produces a new one. The *recombination operator* takes two or more individuals and combines them to produce a new one. More complex operators could be built from those elementary operators. For instance, it could be a recombination operator followed by a mutation operator, or a mutation operator randomly selected from a set of operators. Some instances of evolutionary algorithms relies solely on mutations operators, whereas others are driven mostly by the recombination. Those operators are completely linked with the data structure of the genotype. The (genotype, variation operators)

pair is commonly referred to as the *representation* used for a given evolutionary algorithm.

The selection operator has a bias towards better solutions, it contributes to the selection pressure. One of the early implementation of such an operator is the *roulette wheel* selection, where the probability to select a given individual is proportional to its fitness. Such a selection operator is however problematic, as the scale of the fitness is somewhat arbitrary – and different scaling for the same fitness then leads to different algorithm behaviors. More popular schemes nowadays are solely based on fitness comparisons: *ranking* first sorts the population by decreasing fitness, and then performs a roulette wheel selection on the ranks of the individuals. Another scheme is the *tournament selection*, where N individuals are chosen among the population with a uniform probability, and the one with the best fitness amongst these N is returned. Larger values of N lead to larger selection pressure, thus providing a tunable selection operator. All details, as well as many more selection operators, can be found for instance in [28].

The replacement operator also has an influence on the selection pressure. For instance, *elitist* replacement operators always preserve some of the best individuals from one generation to the next, ensuring that the best value of the fitness along evolution will never decrease. On the other hand, *non elitist* replacement operators can sometimes decrease their current best value, making the search process less likely to be trapped in some local optima. Here it is a non exhaustive list of common replacement operators, that will be evoked further in this thesis (see again [28] for more details).

- *generational* At each generation, the whole population is replaced by the offspring obtained through breeding. This replacement is non-elitist.
- *steady-state* At each generation, a single individual is replaced in the population. A single new individual is built through breeding, and replaces a poorly-performing individual of the population, generally chosen using some reverse-tournament selection. This replacement is elitist.
- $(\mu + \lambda)$ At each generation, μ parents individuals are used to breed λ offspring individuals. The best μ out of the $\mu + \lambda$ parents plus offspring become the next parents. This is an elitist replacement operator.
- (μ, λ) At each generation, μ parents individuals are used to breed λ offspring individuals. The best μ out of the λ offspring become the next parents: this is a non-elitist replacement operator.

1.1.2 Lamarckian and Baldwinian evolutions

For some problems, domain-specific optimizers exist to optimize either some parts of a genotype, or the whole genotype. For instance, we might want to optimize ship hull shapes, represented as the control points of spline surfaces. An evolutionary algorithm would generate rough guesses of ship hull shapes.

At each generation, each newly generated candidate shape would be optimized with a domain specific optimizer, and injected in the next generation population. This hybridization of an evolutionary algorithm with a domain specific optimization algorithm is called *Lamarckian evolution*. One potential advantage is the ability to cope with rough variation operators, while the specialized optimizer is in charge of the fine tuning of the solution.

However, Lamarckian evolution approach requires a specialized optimizer that works on the genotype. Such an optimizer might not exists, but on the other hand, a specialized optimizer that acts on the phenotype might exist: it is then possible, before the evaluation a candidate phenotype, to optimize this phenotype using the specialized optimizer. Furthermore, even when a specialized optimizer exists in the genotype space, giving the fitness of the optimized genotype to the offspring while keeping the non-optimized genotype in the population might prove a better choice than doing Lamarkian evolution [46]. Such approaches are based on the so-called *Baldwin effect* [118] and avoid a too fast convergence toward the first encountered good local optimum.

On the practical side, both the Lamarkian and the Baldwin approaches give the same advantage: the ability to work with rough variation operators while relying on the specific optimizer for the fine tuning.

1.1.3 On the design of an evolutionary algorithm

Like others *meta-heuristics*, Evolutionary Algorithms do not rely on strong hypothesises about the problem to solve. If not much is known about the problem, this is a strength. Other optimization algorithms like the Conjugate Gradient Method or the Simplex Algorithm put very tight constraints as to how a problem must be formulated. But on the other hand, they are usually much more efficient when those constraints are met. Thus, while setting up an Evolutionary Algorithm to solve a particular problem is very easy, a first naive choice of representation and variation operators is likely to end up in a failure. This is why there is no single really general, and versatile Evolutionary Algorithm. Indeed, an important part of the literature devises specialized, domain specific representations and variations operators.

We can formulate what looks like an ideal representation with its associated variation operators. A more detailed detailed view is given in the excellent tutorial ([108]). Sadly, there is no explicit means of achieving clues those properties, but they act as goals when using Evolutionary Algorithms for Evolutionary Design.

- *Coverage of the initialization operator* An ideal initialization operator should provide a uniform coverage of the genotype space without breaking the constraints of the phenotype space. The idea is to reduce as much as possible the likelihood of starting the search trapped in a bad region. Even for simple data structures, as bit vectors, such a coverage merits deep studies, as in [60]. For more complex structures like trees, generating a

uniform coverage of the genotype space is really difficult if at all possible in the case of unbounded spaces ([53], and chapter 5, section 1 of [100]).

- *Ergodicity of the variation operators* The variation operators should be able to reach any point of the genotype space from any other point of the genotype space, in a finite number of applications. Excluding some genotypes means excluding potential solutions.
- *Isotropy of the variation operators* Without any selection pressure (i.e. when selection amounts to random selection), the variation operators should not be biased toward any particular region of the search space. Though, a tunable anisotropy of the mutation operator can be fruitfully used by an adaptive strategy (see [83] for a comprehensive review on this issue).
- *Locality of the mutations* Points of the genotype space transformed by a mutation operators should be close in fitness value. Ideally the mutation locality should be tunable: weak mutations are likely to generate mutated solutions with very close fitnesses, strong mutations are likely to generate solutions with larger fitnesses difference. All stochastic search algorithms rest on this locality hypothesis. Indeed, if this locality is not present, the search process degenerates in random enumeration of the search space ([107]). Again, tunable mutation operators can be fruitfully used by an adaptive strategy. A non-tunable operator will both restrain the quality of the solution (because it can't fine tune solutions under a given level), and the speed of the search (because it can't switch to larger big search steps).

Compared to more informed approaches, like the algorithms relying on gradients, Evolutionary Algorithms scale badly with increasing problem dimensions: they require larger population and longer times to converge.. Furthermore, direct descriptions of physical objects are often of high dimensionality. Hence, proposing some compressed representation is essential in practice. However, before discussing possible representations for Evolutionary Design in Section 1.3, we will first detail some more specific properties of such representations arising from to the particular context of Evolutionary Design.

1.2 Simulation and evaluation

The precise definition of Evolutionary Design for the purpose of this thesis is the combination of Evolutionary Algorithms with numerical simulations, in order to find and optimize designs of physical constructions, hopefully new and creative ones. As described above, Evolutionary Algorithms can be used with any kind of representation. But in practice, the design of the representation and its associated variation operators requires a lot of work. Evolutionary Design, as defined here, is quite general. Still, we believe that working with physical simulations has a lot of implications, influencing the design of the search algorithm.

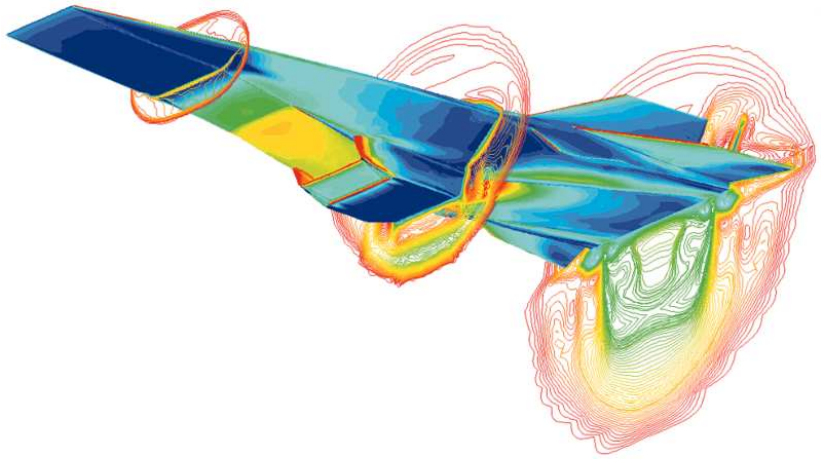


Figure 1.3: A visualisation of a fluid dynamics simulation, for a X-43A vessel flying at Mach 7. It is based on an approximation of the Navier-Stockes equations, solved by the Finite Volume Method.

Physical phenomena are usually described in terms of differential equations (or Partial Differential Equations – PDEs), describing the evolutions of physical quantities both in time and space. For instance, Maxwell's equations describe the evolution of the electro-magnetic field generated by a given, possibly itself evolving, charge and current repartition. Navier-Stokes equations describe how a fluid evolves through time. Indeed, many crucial phenomena for engineering are described by a system of PDEs. PDEs give an inherent local definition of a phenomenon: they describe how the situation at one point is evolving, just as a consequence of its neighbourhood states. Most models from physics, mechanics, chemistry are also non-linear, meaning that analytical solutions of those equations are not accessible in general. Numerical simulations are then the only way to evaluate a global behaviour for a given scenario, like knowing the drag of a plane wing in a stable, linear air flow (see figure 1.3 for an example of such simulation).

Most of the numerical solving algorithms use a discretization of space and time, like mesh models of shapes to solve differential equations by Finite Element Analysis. One implication for evolutionary design is that candidate solutions should be either represented in the representation used by the simulation, or use a transformation to fit that representation. Mesh models, like others, have constraints on their own, meaning that either some candidate solutions might not be suitable for a simulation, or must be repaired to meet the simulator constraints. Thus, using a simulator in order to evaluate candidates solutions adds some constraints to the representation and its associated variation operators.

Very often, however, it is more practical, when evaluating a genotype, to first transform it to another representation, its phenotype (see Section 1.1). For instance, a plane shape can be represented as a spline surface. The genotype would be the vector of the control points coordinates, and the phenotype, the surface itself. The simulators can use a discrete form of that surface, like a Delaunay triangulation, with all the necessary constraints for the simulation to take place. This raises an issue: all the possible phenotypes might not be usable by a simulator. Simulation codes are usually built on a trade-off between accuracy, efficiency, and numerical stability [5] Since Evolutionary Algorithms are stochastic, they might produce candidate solutions whose phenotypes will simply crash the simulator, or make it provide nonsense results: the fitness of those problematic candidate solutions will be nonsense as well. Indeed, an Evolutionary Algorithm can actually *cheat*: producing candidates that are not correctly simulated but have a high fitness by chance [6]. Such cases might be handled by checking the behavior of a simulation and discarding the candidates that lead to the abnormal behaviors. An other way is to process each phenotype before feeding it to the simulator, to ensure it will behave well. This is called *phenotype repair*, [5] chapter 9.

1.2.1 Design of an evaluation function

Once we are sure that any possible candidate solution is not going to break the simulation code, are we done? Simulating is not evaluating. Simulation just provides meaningful quantities about how adequate is a design for a given test scenario. The work of writing an algorithm to evaluate a simulation is itself a tricky job. For instance, the weight of a chair is trivial to compute, its behavior under a load is accessible, but how can we possibly compute its comfort? We could simulate the human as a ragdoll, and check criterion about the general body posture. Such a function would be non-derivable, non-continuous and very noisy. As a consequence, whereas the simulations used for a design evaluation are somewhat predictable and well-behaved, few assumptions can be made about the functions which evaluate designs by means of simulation. This is true for many engineering problems: a design space is open-ended, of mixed nature, and has rather awkward constraints. In this thesis, we assume that for Evolutionary Design, we cannot make many assumption regarding the evaluation function.

When exploring a design space, we might not have a clear idea on how to compare different solutions to a problem. We might have multiple, conflicting criterion of comparison. When designing a bridge of brick, we want to have a bridge as solid as possible. But does a wall 10 meters thick make a good

[5] For instance, when integrating the Newton's equations of motion, Verlet methods provide fast and approximate simulations, with a high numerical stability. In the other end, Runge-Kutta methods provide accurate results, but with a higher computational effort and with a limited numerical stability.

[6] For instance, one of the author's systems evolved teleporting legged robots while optimizing walking gaits. Some parameters settings were able to make the legs of the robots moving very fast, beyond what the simulator was able to handle. The result was a catastrophic propagation of numerical errors, leading to the robots teleportation... See also [82].

bridge? Another criterion is the quantity of bricks used by the bridge. There are two criterion, the bridge solidness and the number of bricks, which seem conflicting. One might aggregate two (or more) evaluation functions into a single one: weighted sum of the functions, product of the functions, etc. Doing such an aggregation is equivalent to taking a decision on the objectives relationship: the optimum reached is dependent on the aggregation of the objective functions. In a design context, this can be troublesome, since we don't want such a choice *a priori*, but *a posteriori*.

An idea in such situations is to search for the *Pareto-optimal* solutions: a set of solutions that do not *dominate* each others, called the *Pareto frontier*. In a multi-objective context, a solution A *dominates* a solution B if A is better than B for all the optimality criterions considered. Multi-objective evolutionary algorithms are able to retrieve a *Pareto frontier* [20, 16]. To allow sound *a posteriori* decisions of what are interesting solutions to a design problem, it is essential to have a diversified Pareto frontier. A diversified Pareto frontier contains solutions expressing a lot of different compromises between the various objectives.

1.2.2 The usage of simulation

By using a simulation simply to provide a scalar fitness measure to a candidate solution, we might under-use the information provided by that simulation. As stated above, most of the models in physics are local, thus the simulations provides local information, both in space and time. Such local information can greatly improves the quality of an Evolutionary Design approach. For instance, when optimizing block stacks, knowing the local reactions forces between blocks might allow for better adjustments than just relying on one single global scalar value. Such adjustments would be much harder to achieve just with random perturbations of the block positions. A higher balanced block stack is more likely to collapse under random perturbations, making an optimization based on non-informed, random variation of the blocks positions, completely crippled. This is true in many design problems, where constraints (like the balance of a block stack) are easily broken under random variation.

Using the local information provided by a simulation can indeed completely change the complexity class of a problem. Here it is a thought experiment, again with blocks stacks, though not aiming at maximizing the overhang[7]. We want to optimize blocks stacks that rest on an immersed ground. The objectives is to maximize the horizontal span of the stack above water. If a stack is fully immersed, then it fitness is 0, since it has no horizontal span above the water. We can use a simple blueprint representation of a block stack, like a list of the horizontal positions of the blocks. No matter how smart the evolutionary algorithm used to optimize the blueprints, bootstrapping evolution can become an issue: if the initial population are just underwater blocks stacks, it can not

[7]This problem was set up by the author for the 2007 Challenge of the French *Journées Evolutionnaires Trimestrielles*

do better than random search. From the algorithm point of view, all the fully underwater stacks are the same, the ones with a null fitness. When, by luck, a candidate solution has at least a single block above the water, the evolutionary algorithm can do something better than trying just random designs. A quite disturbing consequence of such an approach is that the optimization cost depends on the water level. Higher water levels means longer times to find by luck a stack above the water.

Such a problem might seems artificial, and it is. Here, the problem comes from a rather inadequate fitness function, that provides too little information for the task. Here, the lacking informations is obvious (knowing if a block is underwater or not). But it might be not obvious which information is missing for more complex problems involving non-intuitive physics, like in antenna design or aerodynamic shape design. A possible approach is to optimize a *building process*, rather than an actual design. The building process would be dependent on the local information provided by the simulation. For instance, in the stacks blueprints, we can introduce building instructions that are aware of their environment, like *"execute the next instruction if the last added brick was underwater, else skip it"*. Thus, the time spent by the optimizer to generate just underwater stacks is the time to find, by luck, a sequence of instructions that pile-up blocks until the block stack is above the water. If such a sequence is short, then it will be quickly found by a random search, totally independently from the water lever.

In general, working with *building processes* rather than buildings can provides several advantages. The one we mentioned through a thought experiment is the move from a potentially intractable problem to a much easier problem. An actual building gives an answer to a single problem, whereas a process aware of the environment might express the same things in different environments. Evolutionary Design approaches that use environment information are termed *environment-aware*.

1.3 Representations for Evolutionary Design

Evolutionary Design rests on Evolutionary Algorithms, and thus relies on the design of a genotype representation. Representations for Evolutionary Design have to meet constructibility constraints, and even "simulability" constraints. Furthermore, a rather elementary thought experiment leads us to consider the coupling of the information provided by a numerical simulation with the genotype representation. We introduce here various concepts used to design representations for Evolutionary Design.

1.3.1 Block stacks

Again, we will build our discussion around block stacks. It is a classical problem, a playground for theoretical mechanics from the 19th century. Everybody knows what a block is, is likely to have played with blocks at a time, or could readily

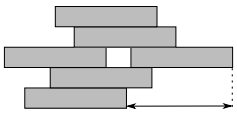


Figure 1.4: Overhang of blocks stack

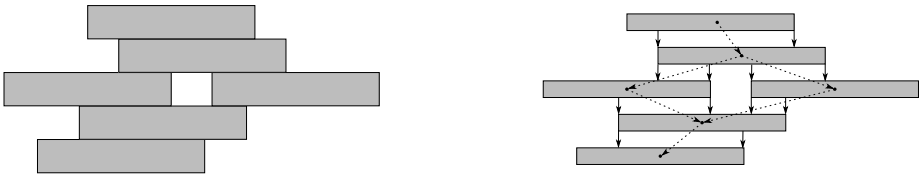


Figure 1.5: Balance of a block stack and forces network

do so. The physical model behind blocks stacking is simple to understand and compute, so little to none technical details are likely to blur our arguments. Block stacking make a perfect ground to exemplify the abstract concepts behind Evolutionary Design.

The classical stack design problem is the *overhang problem*. With a given number of blocks, the objective is to find a stack that stands on the ground with a single block, and that maximizes the overhang of the stack on the right. The stack should be well-balanced. The balance of a stack, though not very complicated to compute, is precisely defined in Appendix B. The main point to keep in mind is that the reaction forces of a balanced block stack are the solution of a linear programming problem. Those reaction forces map to the adjacency relationships between blocks, forming what we call a *force network* (see figure 1.5).

An additional reason to focus on block stacks is the remarkable work of Mike Paterson and Uri Zwick, [94, 93]. Their work provides all we need to use blocks stacks as a benchmark problem. They found [8] the best known overhang for stacks of moderate size, up to 30 blocks. They provided the proof that the maximum overhang achievable with n blocks of unit length is $6n^{\frac{1}{3}}$. The proof is not a constructive one, it does not provide an algorithm that builds balanced stacks with optimal overhang. Nevertheless, the authors of this proof show an example of an algorithm, the Paterson-Zwick construction, which has the asymptotically optimal overhang, $O(n^{\frac{1}{3}})$. This construction produces the so-called *parabolic* stacks (figure 1.6).

In [40], J. F. Hall introduced *spinal stacks*. Though they have an asymptotic overhang of $\log n + O(1)$ (so they are not asymptotically optimal), they are very close to the optimal stacks for moderate amount of blocks. Indeed, for each spinal stacks with a given (small) number of blocks, only small adjustments of the blocks positions have to be made to obtain the optimal stack. The forces at

[8] Paterson et al. used exhaustive search for the stack topology, and for each topology, they used numerical optimization for the blocks positions

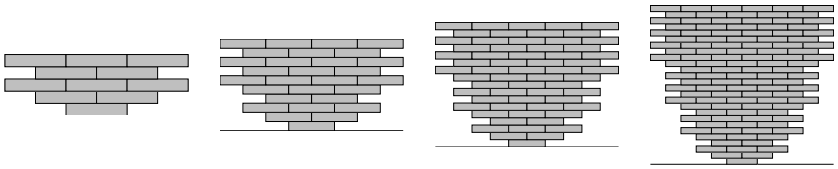


Figure 1.6: Parabolic stacks

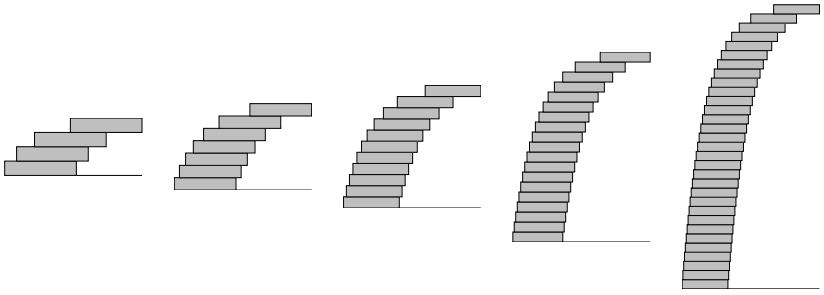


Figure 1.7: Harmonic stacks

the contact points, computed to check if a stack is balanced, are crucial to make those adjustments. Such constructions were breakthroughs from the classical construction, the *harmonic stack* (figure 1.7). A n blocks harmonic stack is built as follows:

- The first block is dropped at position 0
- The second block is dropped at a block length times the Nth term of the harmonic serie.
- The Ith block is dropped at a block length times the Nth - i term of the harmonic serie

Such a construction has an overhang of $\frac{ln(n)}{2}$.

The optimal overhang problem is an interesting benchmark for comparing different approaches of Evolutionary Design. It exhibits many properties of the typical problems in Evolutionary Design:

- The design space is not a continuous vector space, but something with both discrete and continuous components.
- The evaluation function is not derivable nor continuous
- The evaluation function is not convex and is irregular
- The design space has non-linear constraints
- Whereas the physical model behind the evaluation function is simple and easy to compute, it provide little to no clues about the optimal designs

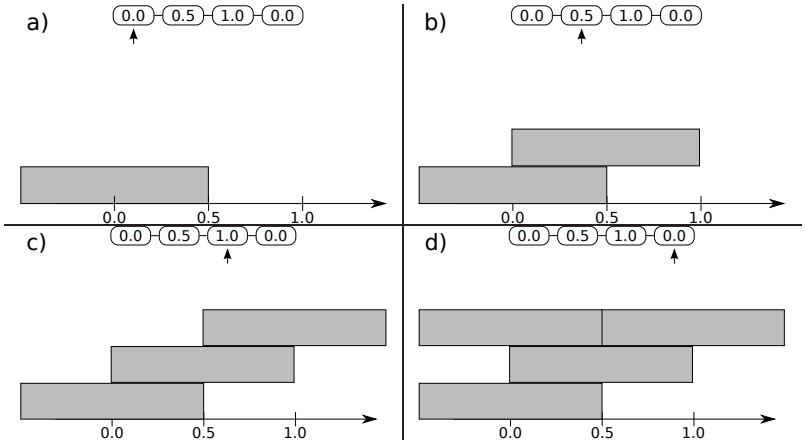


Figure 1.8: Constructive representation of a block stack.

1.3.2 Explicit and implicit representations

In this work, we call *explicit representation*, any representation for which there exists a computable function that takes as argument the size of the genotype and returns a bound on the size of the corresponding phenotype. If no such computable function exists, the representation is an *implicit representation*. A simple example of an explicit representation for a block stack would be the list of the horizontal positions of its blocks centers. This is a constructive representation: a first brick is dropped by using the first element of the list, and so on, as shown on figure 1.8. This way, only stacks with non-overlapping blocks are represented. Still, the unbalanced stacks are also represented. The time to decode a list of positions is then easily predicted from the size of the list.

Explicit representations are the most straightforward way to optimize designs. An explicit representation typically includes all of the traits that define a candidate design. In almost all cases, the size of the phenotype is at most an affine function of the genotype size. Then, because of the difficulty of search in high dimensions, moderates amount of traits from a design family will be practically explored. Nevertheless, in some problems (antennae designs [76], some mechanical [42, 41] and electronically [81, 79, 80] engineering problems), a sub-optimal answer might well be as good as, or better than, the results produced by any team of human engineers, as will be discussed in next chapter. In such cases, explicit representations are good enough. Also, explicit representations could provide practical insights to move forward to more complicated representations. We also must underline that some very high dimensional problems are much easier to solve than low dimensional problems. But then it is very likely that convex optimization techniques would be a better choice than evolutionary algorithms.

Explicit representations should not be stated as dumb or naive: having a

direct relation between genotype and phenotype sizes encompasses very sophisticated mapping schemes. Such sophistication allows to include domain knowledge, such as phenotype reparation. Returning a valid phenotype for any random genotype is often difficult, so adding a phenotype repair stage relieves the search algorithm from coping with non feasible regions. Also, as will be shown in chapter 2, Section 2.1, explicit representations can lead to very nice genotype compressions schemes, and hence to well performing Evolutionary Algorithms setup.

1.3.3 Modularity

Compression and randomness are tightly related concepts: in information theory, a random sequence is defined as a sequence that does not admit any shorter description than itself. Thus, a random sequence is not *compressible*. For a given representation, if we do not take some precautions, a random genotype will code for a very random looking design. For instance, we can code planes shapes as parametrized surfaces, as the ones routinely used in Computer Graphics. Random parameters will give most of the time very random looking shapes, very likely to perform awfully bad in regards to the fitness at hand. Only very sparse and localized regions of those parametrized surfaces code for interesting shapes.

The problem is hence to find representations that code only for insightful shapes. Of course, doing it would mean that we already solved the design problem. So the best we can do is to bias the representation toward what we believe to be sound designs. There is at least a very common and long known bias: *modularity*. Complex human-designed objects, as well as natural beings morphologies, feature symmetries, repetitions, and self-similarities. Restricting the exploration to the designs with repetitions and symmetries, is close to the way humans usually explore designs. When evaluating a candidate genotype, that genotype is seen as a program for a generator, generating a candidate design, the phenotype. For instance, plane shape genotypes would be symbols strings, which when they are evaluated, are interpreted as a program that generate the plane shape. If we have some recursions instructions, we can expect very complex shapes even with very short genotypes. The size of the phenotype from the genotype could no longer be inferred from the sole size of the genotypes.

Out of the compression argument, the *modularity* might provide a further advantage. A single beneficial mutation on the genotype part that codes for a phenotype part, will have an effect on each instance of that part. If identical phenotype parts are coded multiple times in the genotype, multiple beneficial mutations are needed to have the same improvement on each part. A compressed genotype structures the search in the phenotype space. This idea is well studied in the seminal work of Herbert Simon [110]. In a thought experiment, before the advent of the Evolutionary Algorithms, Herbert Simon compares two watchmakers, Tempus and Hora. Hora has a modular approach to build watches, designing them with modules, sub-modules, and so on. Tempus build his watches parts after parts. With elementary mathematics Herbert Simon showed that Hora was a much more reliable worker, much less disturbed by

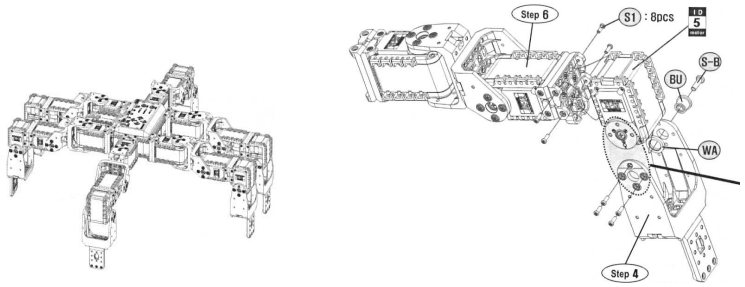


Figure 1.9: Example of a modular robot design.The legs are the same up to a mirror symmetry. The leg are themselves made of parts from a reduced element set. Thus, a description of such robot can be very short, such as a sequence of assembling operations.

random perturbations in his work. Hora tends to *scale* better: he tends to confront to higher complex watch designs ways better than Tempus. The capacity to confront higher complexity of phenotypes by moderate increases of the genotype size is the *scalability* of a representation.

So far, we have been very evasive about what is exactly a module. Indeed, we are not aware of a single definition, nor one that the community agrees upon ... Is it a phenotype part repeated multiple times? But this is too much restrictive, as for instance a robot leg is present twice in the robot body, but the two legs are only identical up to a mirror symmetry. Thus it might be better to have a more functional definition of a module. John Rieffel, in his PhD thesis [105], studying Evolutionary Fabrication, proposed an interesting definition of a module: Any part of a genotype could somehow code for a module, what is a module is rather subjective. But there exist good and bad modules. Thus, still according to John Rieffel, a good module is a part of the genotype that codes for several parts of the phenotype. Those phenotype parts exhibit a unimodal behavior with a very low variance even in presence of noise or in different environmental conditions. John Rieffel provided experimental evidence about how good modules emerge. He optimized through Evolutionary Algorithms small programs that coded for 2d objects made of blocks stuck together. But when evaluating a candidate program, the program was executed with some random errors. Higher levels of noise led to highly compact programs, relying highly on recursion to build 2d objects. Also, programs optimized with higher levels of noise triggered fitter block constructions, and more reliable building processes. Those experiments and the results are described in the chapter 5 of John Rieffel's PhD thesis. This is clearly related to Herbert Simon's tale, with his watchmakers and the high importance of the resilience to randomness. Hence we will stick to John Rieffel's definition of modules, which is representation independent and built on an analysis of actual experiments. Other's definitions are, according to us, either blurry, or dependent on some representations, never as insightful.

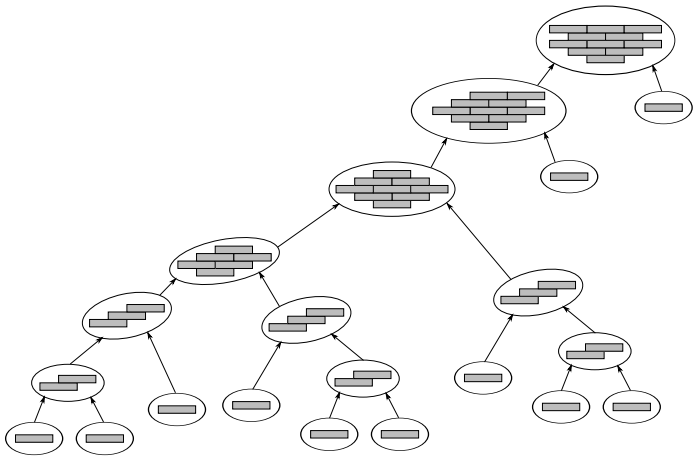


Figure 1.10: An assembling plan as a tree structure, for a parabolic stack

1.3.4 Tree structures

One idea in order to have *modules* in a representation is to use hierarchical representations of a design: some *top* parts are built upon *bottom* parts reused several times. Such an idea of modularity maps perfectly with the ubiquitous tree data structure, as illustrated by figure 1.10.

This is indeed what was done by John Koza et al. in the scope of the *Genetic Programming* [69]. Genetic Programming is a general method to setup an evolutionary algorithm that optimizes symbolic trees, introduced by Michael Cramer [18] and independently by Jurgen Schmidhuber et al. [24]. If the leaves of a tree are elementary parts, and the nodes are assembling operations, then this tree can be seen as an assembly plan. However, in practice, optimizing trees through evolutionary algorithms does not lead to the emergence of *modules* ([100], chapter 6, section 1).

John Koza proposed ([70]) to see a single genotype as a set of trees, and to consider special leaves that are references to one of the trees present in the genotype. Those special leaves are called Automatically Defined Functions (*ADFs*). Figure 1.11 illustrates what the tree from figure 1.10 is with the ADF representation. With ADFs, the interpretation of a tree can generate phenotypes of different sizes from genotypes of the same size, depending how frequently a sub-tree is reused.

ADFs are a very direct translation of the Tempus and Hora fable, and as in the fable, we might except much better results from the Hora approach. An equivalent approach to trees to ADFs is to use Directed Acyclic Graphs (*DAGs*). As shown in figure 1.12, a DAG is a graph similar to a tree, except that a node can have multiple parents. By simply replacing the special leaves used with ADFs by edges, then we obtain a DAG.

Is Genetic Programming with ADFs, as introduced by Koza, likely to help

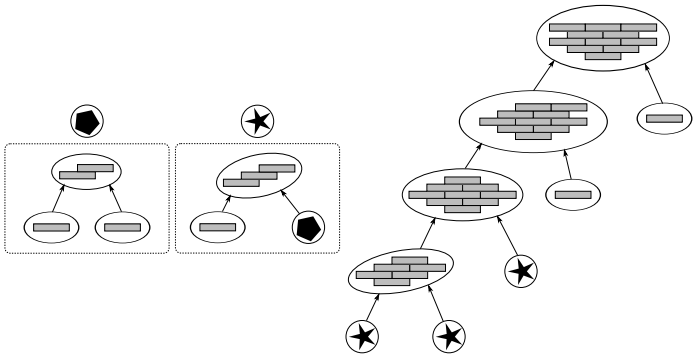


Figure 1.11: An assembling plan as a tree with ADFs, for a parabolic stack

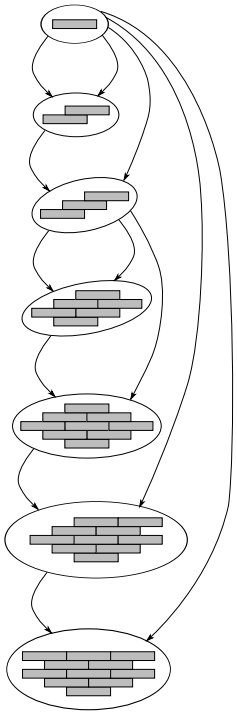


Figure 1.12: An assembling plan as an explicit DAG, for a parabolic stack

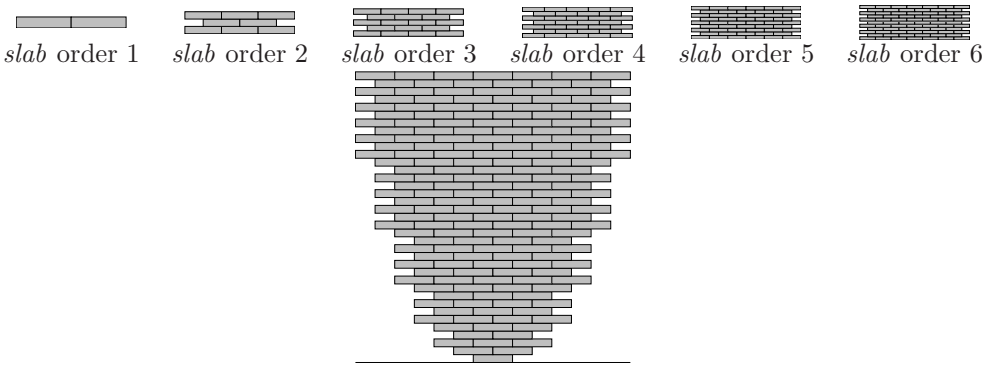


Figure 1.13: A modular decomposition of a parabolic stack. A same module is reused, the *slab*, but each reuse instance is of a different scale. Thus, a pure reuse of a same module is not able to capture the modularity of the parabolic stack

for Evolutionary Design problems, like the block stacks overhang problem? If we look at the parabolic stack example, a small 3 blocks pattern is replicated three times. Is it a scalable representation of parabolic stack? When looking closer at the parabolic stacks (see figure 1.13), the simple replication of a part does not capture the regularity of the parabolic stacks. A more complex part reuse mechanism is needed to represent parabolic stacks in a compact way that could scale up well, accounting for the fact that the same part is reused, but at different size orders. Without this more sophisticated reuse mechanism, greater parabolic stacks would require more ADFs, producing bigger genotypes, and not scaling up nicely. In next subsection, we present a class of more complex reusable mechanisms, powerful enough for instance to capture the modularity of the parabolic stacks.

1.3.5 Grammars and rewriting systems

ADFs can be seen as an elementary tree rewriting system: a special leaf is replaced by a complete tree. Why then not use more sophisticated rewriting systems? Although introduced for plant shape growth modelisation by Aristid Lindenmayer [102], L-systems are rewriting systems that are general enough to describe many kinds of entities with an underlying strong regularity: the Cantor dust, Penrose tiling, Sierpinski triangle, the Koch snowflake, etc... L-systems can be used to produce very complex strings of symbols from a little set of rules. In particular, the string produced by an L-system can be interpreted as an assembly plan for some design. For instance, for the blocks stacking problem, we can use an L-system to produce the sequence of dropping positions.

The most basic L-systems are the D0L-systems. A D0L-system is a triplet $< V, w, P >$, where:

- V is an alphabet, with $V^\star$ the set of all the words over V, V^+ the set of

all the non-empty words over V.

- w is the *axiom*, an element from V^+
- P is a finite set of elements from $V \times V^\star$, the *productions*. A production is written as $a \Rightarrow X$, where a is from V, and X from $V^\star$.

While Lindenmayer also defined non-deterministic L-system, here we only consider deterministic L-systems and assume that, for each a from V, there is exactly one X from $V\star$ such that $a \Rightarrow X$. Here is an example of a D0L-system with the $\{A, B\}$ alphabet. Note that the production in the form $a \Rightarrow a$, the identity production is assumed to belong to the set of productions.

$$\begin{aligned} w &: A \\ p_1 &: A \Rightarrow B \\ p_2 &: B \Rightarrow AB \end{aligned}$$

Then, from a D0L-system, a sequence of strings can be produced, called the *developmental sequence*. The first term of sequence is the word w. A further term of the sequence is built from the previous term, by replacing each symbol of the string by a word, according to the productions set. Here are the first terms of the developmental sequence by the D0L-system above. Note that the length of the strings produced corresponds to the Fibonacci sequence.

$$\begin{gathered} A \\ B \\ AB \\ BAB \\ ABBAB \\ BABABBAB \\ ABBABBABABBAB \\ BABABBABABBABBABABBAB \\ \dots \end{gathered}$$

After computing several terms of the developmental sequence, we can interpret the string produced as a building plan. Which term of sequence should be picked to interpret it? For some productions set, the developmental sequence reaches a fixed point: this is when a term is transformed into itself by the production set. The fixed point of a developmental sequence would make a natural candidate as a definition of the result of the sequence. But the existence of a fixed point is not verified for all the productions sets, and an L-system can also lead to a limit cycle, or produce an ever growing string. Then it requires artificial ad-hoc stopping criteria.

D0L-systems can produce long sequences with few productions and a little alphabets. There is no way to predict the size of the string produced by a given D0L-system just from the size of the $< V, w, P >$ triplet members: D0L-systems are indirect representations of strings. However, using D0L-systems for the block stacking problem is not possible, because block positions are continuous. Introducing a bit of continuity into the D0L-systems would make them more useful to represent designs, which often contain continuous quantities. This is exactly what the parametric D0L-systems are made for. The words used by the parametric D0L-systems can accept numerical parameters. The production rules are also extended, they have predicates to decide in which case they will be applied when producing the developmental sequence. Here it is an example of a parametric D0L-system that produce the harmonic stack with n blocks:

$$\begin{aligned} \text{start}(n) &\Rightarrow \text{harmonic}(n, 1, 0) \\ \text{harmonic}(n, i, x) \colon i \leq n &\Rightarrow \text{harmonic}(n, i+1, x + \frac{1}{i+1})\text{drop}(-x) \\ \text{harmonic}(n, i, x) \colon i > n &\Rightarrow \end{aligned}$$

The parametric word drop(x) is the block dropping command with the drop position as parameter. start(n) is the axiom of the parametric D0L-system that code for a n blocks harmonic stack. harmonic(n, i, x) is a non terminal word, as a recursive definition of the harmonic stack.The corresponding developmental sequence, with $n = 6$, is:

$$\begin{gathered} \text{start}(6) \\ \text{harmonic}(6, 1, 0) \\ \text{harmonic}(6, 2, 0.5)\text{drop}(0) \\ \text{harmonic}(6, 3, 0.8333)\text{drop}(-0.5)\text{drop}(0) \\ \text{harmonic}(6, 4, 1.0833)\text{drop}(-0.8333)\text{drop}(-0.5)\text{drop}(0) \\ \text{harmonic}(6, 5, 1.2833)\text{drop}(-1.0833)\text{drop}(-0.8333)\text{drop}(-0.5)\text{drop}(0) \\ \text{harmonic}(6, 6, 1.4499)\text{drop}(-1.2833)\text{drop}(-1.0833)\text{drop}(-0.8333)\text{drop}(-0.5)\text{drop}(0) \\ \text{harmonic}(6, 7, 1.5928)\text{drop}(-1.4499)\text{drop}(-1.2833)\text{drop}(-1.0833)\text{drop}(-0.8333)\text{drop}(-0.5)\text{drop}(0) \\ \text{drop}(-1.4499)\text{drop}(-1.2833)\text{drop}(-1.0833)\text{drop}(-0.8333)\text{drop}(-0.5)\text{drop}(0) \end{gathered}$$

We can note that the notion of module is more subtly enforced in the parametrized L-systems than in tree representations, like DAGs. In the parabolic stacks case, tree based representations are able to encode a decomposition of the parabolic design, but fail to represent correctly that a parabolic stack is just made of piled-up rectangular chunks of blocks of increasing size. Though modular, a tree based representation will not scale: parabolic stacks of increasing order will need bigger trees. In contrast, our L-system based solution remains of constant size with parabolic stacks of increasing order: we found a *scalable*

representation for parabolic stacks, thanks to the sophisticated L-system recursion.

The parametric L-systems can be enriched with an environmental context. Conditions for a rule, and parameters or a word could be quantities from the simulated environment. Then, we have an environment-aware representation (as exposed in Sub-section 1.1.2) able to represent building processes, with the all the advantages implied.

1.3.6 The evolvability issue

With parametric L-systems, we can have both a coupling with the environment and a certain modularity. L-systems works on strings, which makes them quite general, and they were used to discover and optimize real-world designs. Are we done then? What could possibly go wrong? Searching in the space of the L-systems, even with the simplest case of the non parametric D0L-systems, is quite problematic. A single little alteration on a single L-system production is likely to induce huge changes on the developmental sequence. That means that we do not have the locality property (as introduced at Section 1.1.3), each alteration of the genotype being equivalent to a huge jump in the phenotype space, without any clue about a pattern, a structure on those jumps. The topology of the L-systems space and the space of the words strings are not related at all: two close L-systems does not have close developmental sequences. With such a scrambled relationship between genotypic and phenotypic space, any search procedure is unlikely to do better than random search or brute force enumeration. L-systems lack of *evolvability*.

The evolvability issue we underlined is not necessarily a curse on the L-systems based representations. Very different L-systems could encode for very similar genotypes, which is completely in adequacy with our argument. They might be successful if a random candidate genotype is likely to produce a good design. In this case, even a very simple search algorithm will succeed. Because of the poor search space topology preservation properties of L-systems, simple random and greedy search algorithms will perform as well as sophisticated one.

1.3.7 Cellular representations

Cellular representations are another perspective on the way to define an object as a building process. Previously introduced representations have an intrinsic *bottom-up* and *explicit* nature. The structure of the building process is explicitly mapped to the structure of those genotypes. For instance, the branches of an assembling plan coding for a block stack are explicit decomposition of the stack in several sub-stacks. Due to that explicit hierarchy, the mutation operators are very likely to produce completely different structures at each application, breaking the essential principle of the locality of the mutation (see Section 1.1.3).

The idea behind cellular representations is to remove any kind of explicit hierarchy. A building process is carried out by *cells*, identical entities interacting between them, localised in space. A cell has an *internal state* and an *update*

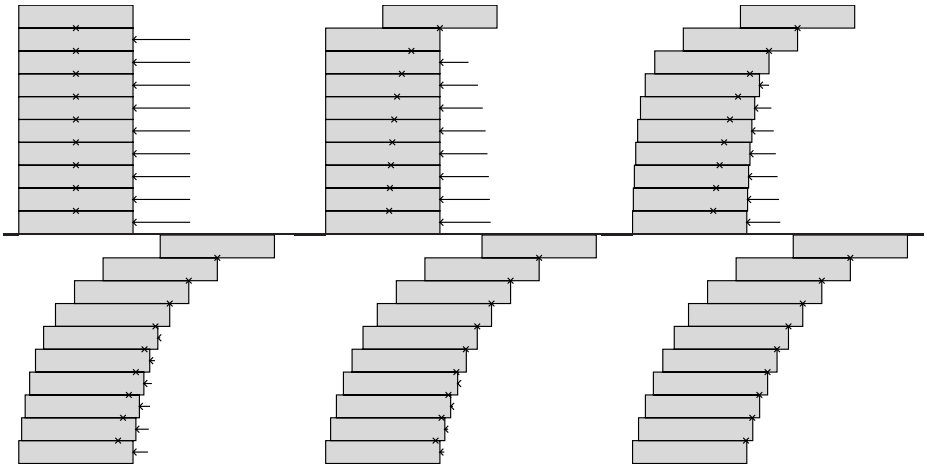


Figure 1.14: An ontogenic definition of the harmonic stack. Each block horizontal position is shifted independently and synchronously, according to the same function. The arrows represent the shift applied to each block. The crosses represents the horizontal position of the gravity center of the blocks over a given block. The harmonic stack is the steady state of the process.

function, modifying that internal state depending on the states of the surrounding cells, and on the input of the surrounding (simulated) physical environment. Both the update function and the initial state are the same for all cells. Over time, due to the different surroundings of the cells, their states evolve. The cells also share a unique *expression function*, which defines how a cell locally transforms its surrounding physical environment. This way, a closed loop feedback with the (simulated) physic is created. The system can be initialized with a very rough guess of the problem solution, and the cells would iteratively transform that initial guess in an optimal solution.

We will practically illustrate cellular approaches in chapters 4 and 5. Here is however an illustration of this idea, using again the block stacks world and the overhang problem. We allow just one block to be over an other block. In that case, the harmonic stack is the optimal solution for the overhang problem [94]. As a rough initial solution to the overhang problem with n blocks, we simply use the vertical pile of n blocks. The overhang is zero, but at least, such a stack is balanced. To each block, corresponds a cell. A cell has access to the horizontal position of the center of mass of the blocks above it. Furthermore, a cell has the ability to move its associated block, by an horizontal shift. The horizontal shifts are totally independent: shifting one block does not move any other block.

The horizontal position of the block i center is noted x_i. The top block is the block 1 and the bottom block is the block n. The horizontal position of the

center of mass of the blocks *above* the block k is noted g_k.

$$g_k = \frac{1}{k-1} \sum_{i=1}^{k-1} x_i$$

Each cell continues to shift it own block in order to keep the center of mass of the blocks above, as much as possible to right side as following:

$$x_i(t+1) = g_i(t) - \frac{1}{2}$$

$$x_i(t+1) = \left(\frac{1}{i-1} \sum_{j=1}^{i-1} x_j(t) \right) - \frac{1}{2}$$

All the cells are moving simultaneously, by discrete time steps, as shown on figure 1.14. We then iteratively shift the blocks, with just the local knowledge of the horizontal position of the center of mass of the above blocks. It should be noticed that g_k is not defined for the top block, so the top block remains fixed.

This process converges in a finite number of steps to the harmonic stack. Indeed, the harmonic stack is the steady state of that process. In that state, the shift computed for each block is 0. This is a global attractor: from any initial configuration, with any number of blocks, the process will end up to the harmonic stack. This is a compact representation of the harmonic stack: an affine function. This is a scalable representation: the update function is independent from the number of blocks in the stack. This is a robust representation: from any initial stack, as far one block is over one block, the process will generate the harmonic stack. Here, we knew in advance the optimal update function, but we would be able to optimize it: there are a lot of universal function approximators that would serve (Perceptron, Gaussian Processes, etc ...).

As such, cellular representations are nothing else than *cellular automata*, coupled with some physical phenomena of particular interest to the design problem. If we consider continuous time, and an update function described as a set of a differential equations, then we are dealing with an *excitable media*, such as the *reaction-diffusion* systems ([116], a quick introduction to reaction-diffusion is proposed in the Section 4.2, chapter 4). Such systems are able to carry computations over space and time, producing a rich variety of patterns [120, 89]. Those patterns features self-similarity: symmetry, repetition, modularity are emergent properties rather than rules of such systems.

As said above, in a cellular approach, a building process is fully determined by the cells initial states, the update function and the expression function. Then, the genotype should encode such functions. Update functions can be encoded in various ways. Genetic Programming ([69]) is an option, as well as neural networks. Some neural networks have the interesting property that they can be defined by a simple vector of real numbers. The optimization of building processes can be then carried out by well-understood evolutionary algorithms like Evolution Strategies. Thus, we can expect to focus solely on

the cellular representation, without having to design new and specific variation operators. Since each cell has the same update and expression function, we can use theoretically as many cells as we need without any impact on the genotype size.

The opportunity to optimize building processes, within a tight coupling with the environment, and encoded as vector of real numbers, seems a promising idea. With the lack of explicit hierarchy in the cellular representation, and a wise enough choice of function encoding, we might address the evolvability issue that plagues previously devised representations. But cellular representations, as we will see in the next chapter, are somewhat recent. The feasibility and the promises of cellular representations remains to be demonstrated. Very important issues remain open:

- Which update functions are suitable? Is there a universal family of update functions? Or are they strongly application dependent?
- When to stop the iterative update process? How to evaluate the effectiveness of a building process?
- How complex are typical update functions? At which problem size is a cellular representation genotype more compact than an explicit representation genotype?
- Are cellular representations truly evolvable?

1.4 Preview

Those issues will be addressed in this thesis, along the following lines.

Chapter 2 summarizes the state of art of Evolutionary Design, discussing the related work while focusing on exposing the widest range of approaches and schemes.

Chapter 3 devises a rather classical and truly general Evolutionary Design approach applied to a simple yet realistic problem: block stacking. Through an empirical study, we will show that the potential flaws pointed on the previous chapter are indeed met in real-life. We show that the key lies in an extensive exploitation of the knowledge about the physical environment in which the designs are tested.

Chapter 4 details our work on cellular representations, carried out by studying one of its most studied archetype problems: the *embryogenic flags*. We show experimentally that even a relatively naive setup is sufficient to harness the expected properties from a cellular representation. The key aspect of those studies lies in the importance of the stopping criterion used when evaluating a development process. An unwise choice of stopping criterion totally destroys the scalability of the solutions, thus loosing one of the main interesting point of the cellular representations.

Chapter 5 shows an application of our approach to a close-to-real-life problem, that of truss design. Through an experimental protocol, we show that all the expected properties of our approach are effective.

Chapter 6 sketches the future direction of research triggered by our work, and what remains to be done. We advocate, for Evolutionary Design, the use of cellular systems, tightly coupled to physical simulations. We focus on the notion of designs encoded as fixed-point of dynamical systems. We conclude by suggesting a few testbed problems which we think are relevant in order to further study Evolutionary Design.

You have to forget about what other people say; when you're supposed to die, when you're supposed to be lovin'. You have to forget about all these things. You have to go on and be crazy. Craziness is like heaven.

Jimi Hendrix

2 A guided tour of Evolutionary Design

In this chapter, we introduce a selection of Evolutionary Design research. This review aims at good coverage rather than being exhaustivity. As explained in chapter 1, the strength of Evolutionary Algorithms lies in their ability to explore and optimize any kind of representation space. As such, they have been used to solve diverse of design problems, where more traditional engineering methods are of little help. We try to reflect this diversity in our review, especially in the first section, devoted to the explicit representations. The second section introduces implicit representations, where the size of the genotype and the phenotype are not tightly linked. The works presented in the second section are often of a more exploratory nature: the goal is not necessarily to address a precise engineering problem, but to validate ideas on the possible ways to design implicit representations. Then, in the third section, we focus more on cellular representations, where the state of the art is even more exploratory.

2.1 Explicit representations

Explicit representations are those which imply a genotype to phenotype mapping complexity which depends only on the size of the genotype. Such transformations are able to carry domain knowledge and to enforce constraints. By far, such representations are the most represented, and we relate here some of the most representatives one.

2.1.1 Block stacks

Tim Hohm et al. [47] suggest a rather canonical evolutionary algorithm to maximise the overhang of balanced stack with a given number of blocks, as introduced in chapter 1, Section 1.3. Their work is a perfect yet simple example of an Evolutionary Algorithm for design search and optimization.

One of the challenge for the representation of the solutions is to encode only feasible stacks, i.e. with non interpenetrating blocks. Tim Hohm et al. tackle this issue by storing a stack as the list of the horizontal positions of all blocks. Thus, whereas the phenotype is a block stack, the genotype is a list of real values. The actual stack is obtained after an interpretation of this list: the blocks are

dropped one after one, according to the list order. When dropped, the horizontal position of a block is taken from the list, while its vertical position is the lowest one which does not generate intersections with the previously dropped blocks. Such a list of horizontal positions can be seen as a building sequence. This is an idea often met in Evolutionary Design: storing building sequence helps to deal elegantly with constructability issues. Here, the chosen representation can encode all the stacks without intersections, and only them. Note that this is not a one-to-one representation: two different lists of drop positions can code for the same stack.

The initial candidate stacks are taken among the N best stacks from $100N$ stacks, randomly generated according to some heuristic. The mutation operator adds a random variable, drawn according to a zero centered Gaussian distribution, to each position of the list. If the obtained stack is not at equilibrium, a new mutation try is done. When a mutation is applied, in one percent of the cases, just a single block is moved randomly. One-point crossover is used as the crossover operator. For both mutation and crossover, if the offspring is not balanced, new tries are performed until a balanced stack is produced. If 100 tries were not able to produce a balanced stack, then the offspring is simply a copy of the original stack.

This algorithm is able to find a nearly optimal overhang for 20 blocks stacks, and interesting results with 30 blocks, yet not optimal. The crossover operator seemed of little use, since runs using few to no crossover were the most effective. The proposed crossover is built on the assumption that good solutions could be made by combining the upper and lower parts of two stacks. It is clearly a highly optimistic assumption. Also, using a larger population size did not improve the results, it just somewhat reduced the variability between runs.

The proposed algorithm does not scale well, as increasing the computer power provides little to no improvement. The network of forces computed to check the balance of a candidate stack is never exploited to bias variations toward new balanced stacks. Without use of that knowledge, bigger stacks are likely to give mostly unbalanced stacks under blind random variations. Trying 100 times a variation operator is a rather crude way to compensate this lack of knowledge usage. Obviously, it makes large stacks out of reach for the proposed algorithm.

2.1.2 Lego structures

Legos structures could make an ideal playground to tinker with evolutionary design besides block stacks. Funes et al. [34, 33] optimize 2D and 3D static structures made of Lego bricks, toward simple structural objective functionalities: bridges and scaffolds. Only the beam-shaped bricks are employed in their constructions. The structures are represented as trees, where a node represents a brick, and an edge represents a connection between two bricks. The genetic operators are the same as the ones used in standard Genetic Programming [69]. Those operators can produce unfeasible individuals, so feasibility checks are

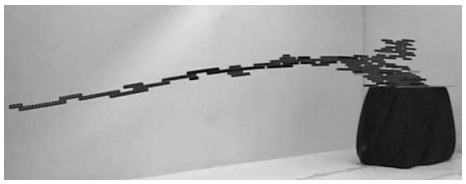


Figure 2.1: A plane Lego construction, optimized for a maximal overhang, from Funes et al. [34, 33].

needed. This is one the first work featuring evolutionary design of real world structures. Figure 2.1 shows one of the best bridge structure obtained by Funes et al.

Still in the Lego structures optimization, Maxim Peysakhov et al. [97, 98], devise an other, more general, representation. While the representation proposed by Pablo Funes is limited to assemblies made of beam parts, the representation devised by Maxim Peysakhov et al. can *potentially* handle the other types of parts as well. They define a graph grammar to represent a subset of the possible assembling operations for a small subset of Lego parts. The nodes of an assembly graph are the elementary parts, whereas the edges specify how two elementary parts are linked. The proposed grammar specifies which are the valid associations of elementary parts. Yet, this grammar allows assemblies with self-penetrating parts, so further feasibility checks are needed when an assembly graph is interpreted. The genotype is made of two parts: a table of nodes and a table of edges. Thus, the search space is a space of graphs, with constraints coming from a user-specified grammar. The exploration of this graph space is done by a Messy Genetic Algorithm [36], where the node and edge tables are coded as bit strings. The structures are optimized toward space-filling objectives: finding structures of minimal volume that fit in a given bounding box.

Pavel Petrovic [96] uses an Evolutionary Algorithm to find Lego layouts that form a given 3D shape, while satisfying a set of construction quality criteria. A Lego layout is represented as an unordered list of tuples *(brick type, position)*. This is an explicit representation of a Lego Layout. Sophisticated heuristics are used to initialize the initial population. No less than seven mutation operators are defined in order to randomly alter a list of Lego parts. A single, problem specific, crossover is also proposed. While being an explicit approach, the genotype to phenotype mapping achieves a lot of smart work, like merging adjacent little parts into equivalent single big parts when possible. Such a mapping is a typical way to embed domain knowledge into a representation.

2.1.3 Antennae

Antennae design is considered notoriously difficult, as an art built over years of experience [72]. Despite the electromagnetic phenomena are well described by the Maxwell's equations, we have little clues about how yo use them to find optimal antennae. Hence antenna design provides a perfect playground for evolu-

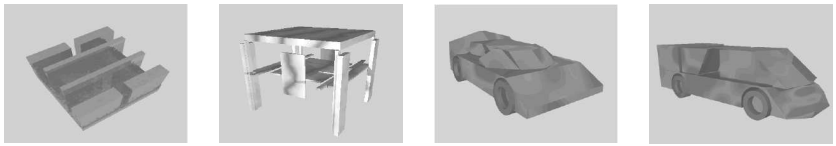


Figure 2.2: Shapes optimized with the generic representation of Peter Bentley [6]. The first shape is an heat sink, the second one is a coffee table, the third one is a sport car, and the last one is a saloon car.

tionary design. Indeed Jason Lohn et al. [74, 76, 75] use parametrized antennae designs, in order to be optimized toward a specific application. Parameters are represented as a real valued vector, optimized by a canonical real-valued genetic algorithm [37]: mutation strength remains fixed and one-point crossover is the main variation operator. An interesting aspect of the approach is the use of *co-evolution*: the fitness function and the antenna designs are evolved in parallel. The intended effect is to smoothly guide the search process from simple problems to the original, difficult problem. Their best optimization results were of superior quality compared to previous fully human designed antennae.

It is unclear whether using more complex genotype representations, coupled to an electromagnetism simulator, would trigger significantly better results. Explicit representations, as the one proposed, seem good enough to find better designs compared to a traditional conception. The choice of the optimization algorithms is not judicious: a state of the art Evolution Strategies would be better. Evolution Strategies are able to adapt there mutation strength, allowing better fine tuning of the parameters.

2.1.4 Shapes

Peter Bentley [6] introduces 3D shape design for several applications: tables, boat hulls, cars, optical prisms, heat sinks, by using the same genetic algorithm coupled to the same representation. He devises an explicit representation to map spatial partitions based on truncated boxes, to real valued vectors. A design is made of given number of *clipped stretched cuboids*. A clipped stretched cuboid is an axis aligned cube, stretched by three independent scale factors along the three axis. Then, the stretched cube is cut by a plane, one of the two parts of the cut cube is used.

One *clipped stretched cuboid* is defined by nine parameters: three for the scale factors, three for the cutting plane normal, and three more for the plane origin, relative to the cube center. Furthermore, the number of cuboids in a genotype is variable: mutations can add or remove cuboids. The cuboids parameters are discretized as six bits values, thus, a genotype is a bit string. The optimization is carried out by a standard genetic algorithm.

The evaluation function is a linear combination of several objectives, where each objective is computed by a module from a module set. Bentley uses, for

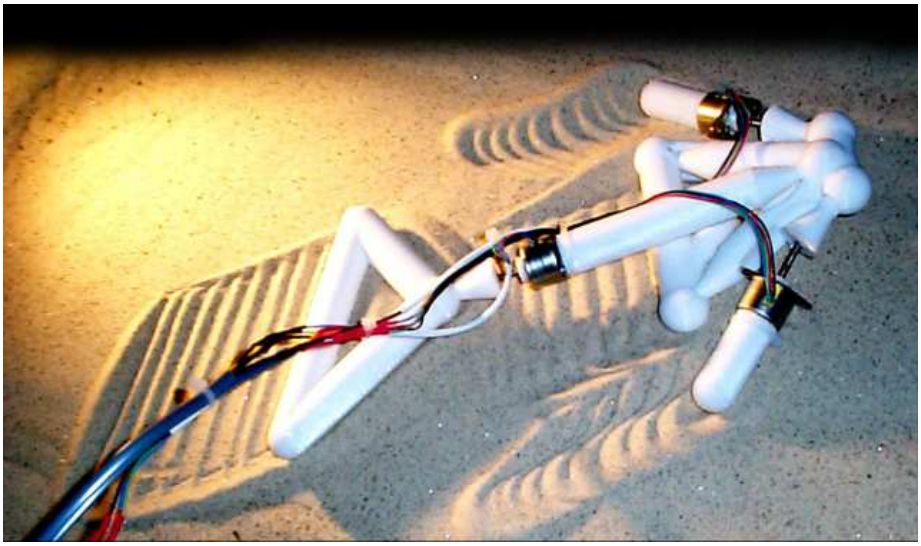


Figure 2.3: One of the best robot designed with the GOLEM project [101].

example, a flow computation module, a center of mass module, a raytracing module, etc. A set of evaluation modules triggers an objective, and thus a design. Examples of different shapes obtained with Bentley's approach are shown figure 2.2.

Bentley's work demonstrates the potential power of Evolutionary Design: rapid and automatic conception of designs is possible with a well-conceived representation. Since the nine parameters of a cuboid are indeed real-valued, it would make more sense to use a real valued vector as a genotype. For such representations, a real-valued Genetic Algorithms, or an Evolution Strategy, are likely to perform better than the bit-string GA used in the above work.

2.1.5 Robots

The GOLEM project by Pollack et al. [101] features the first evolutionary design with both a fully automatic design process and a semi automatic fabrication process. The project aims to be a proof of concept of a new way to conceive robots. The robots are made of plastic beams, joined by ball-and-socket joints, thus making a truss structure. Some of the plastic beams embed a piston actuator, allowing the beam to change it size. The command of the pistons is made by a recurrent, discrete time, neural network, without inputs from sensors. The plastic truss is built by an industrial 3D printer, automating most of the building process. Figure 2.3 illustrates one robot from the GOLEM project.

The robots are represented as graphs. The nodes represent vertices, corresponding to the robot's beam joints. The number of nodes is fixed, it never

changes through the evolutionary run. The edges are the plastic beams of the robots, either fixed bars, or actuated by a piston. An evolutionary algorithm optimize the graphs with operators acting directly on the graph connection matrix: addition or removal of edges, perturbations of the vertices coordinates. The robots are evaluated within a physical simulator, according to a straight locomotion objective.

Families of topologies and associated control emerge, but the system is bound to the simplest solutions. According to Pollack et al. (with the *Golem at home* experiment), whatever the computing power amount used, the evolutionary process stagnates on the same designs. We believe that designs above a given complexity are most of the times completely broken by any random variation. The proposed representation suffers from a scalability issue.

The underlying graph structure of the robots suggests using the more up-to-date *NEAT* algorithm, detailed in Appendix C. This evolutionary algorithm, though originally designed to optimize both parameters and topologies of neural networks, is general enough to work with any kind of labeled graphs. *NEAT* had often featured faster convergence to similar or better fitnesses levels on several problems [113], compared to older, canonical Evolutionary Algorithms working on graphs. As shown later in Section 2.3, a representation devised by Josh Bongard, might be also adopted, avoiding much of the scalability and evolvability issues.

2.1.6 Tensegrity structures

Chandana Paul et al. [95] devise an evolutionary algorithm to discover original tensegrity structures [103]. Tensegrity structures are made of stiff sticks and elastic wires tied together (see figure 2.4), so as when relaxed, they always form the same structure. Such structures are of high interest in architecture and robotics: they can be of arbitrary size and complexity, yet lightweight. Tensegrity structures open wide perspective, for pliable and lightweight structures: antennae, buildings, robots, support structures etc. [88] gives an extensive review of the applications of tensegrity structures.

Sadly, the concept of such structures for a given purpose is still hard [88]. One of the reasons for that comes from the fact that there is no other ways than simulation to predict how a tensegrity structure will relax. Then, this is a field where Evolutionary Algorithms can potentially trigger new insights. Regular structures have been previously explored by relying, for instance, on group theory [17].

Paul et al. use an explicit representation: the vertices of the structure, the connection matrix of the stiff sticks and the elastic wires are all together in a vector of real values. The genotype size is not modified by the mutation and crossover operators. The evaluation of a genotype consists, within a solid physics simulator, in building the structure, and then to relax it. The tensegrity structures are optimized to occupy the highest possible volume once relaxed. The algorithm give original answers, although of little size (up to 12 sticks)

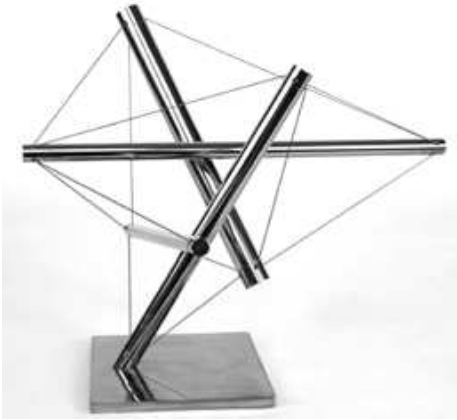


Figure 2.4: An example of an elementary tensegrity structure, by Chandana et al. [95].

compared to the results obtained with the group theory.

Many known tensegrity structures are very regular [103, 88]. Using more structured representations, like L-systems, would probably allow the user to eventually capture such a regularity. This way, much larger structures would be accessible when using an Evolutionary Algorithms. Also, knowing the tensions of elastic wires might helps to adjust the wires length in a more informed way.

2.1.7 Topological Optimum Design

Kane and Schoenauer [61] and Hamda et al. [42, 41] use explicit representations for Topological Optimum Design. The goal is to optimize a shape that meets a set of mechanical constraints (typically, supporting loads while being deformed below a given limit) with minimal weight. The shape to optimize must lie in a design domain, a bounding box for instance. The simulation code used to compute the stress information for a candidate shape uses the Finite Element Method. Hence, the shape requires a discretization, into a square grid with either empty or full squares. The representation used in [61] is then a simple array of bits, one bit per element of a given discretisation. The *Voronoi representation* used in [42, 41] is a set of points, where each point have a label, either *full* or *empty*. The corresponding phenotype is the Voronoi diagram of the set of points (see figure 2.5). The cells with a *full* centroid are considered full of matter, while those with an *empty* centroid are empty. The mutation operators can add points, change the point labels, or slightly perturb the position of points. Furthermore, a specialized crossover operator is defined for this representation. According to our definitions, this is still an explicit representation, as the size of the phenotype (the Voronoi diagram) depends directly on the size of the genotype (the set of labeled points).

On the same benchmark problems for Topological Optimum Design, the

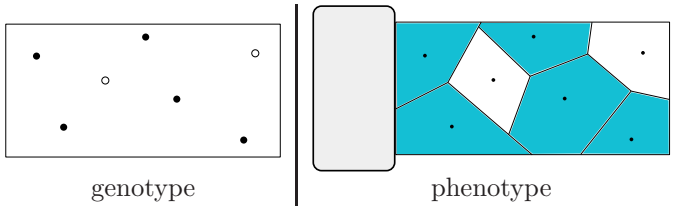


Figure 2.5: In Hamda et al. [42, 41], the genotype is a set of labeled points, either *full* or *empty*, in a given design domain. The phenotype is the Voronoi diagram of those points, with the cells either full or empty depending on the points labels. Here, the phenotype corresponds to a structure that should be able to support a given load.

Voronoi representation performs better than the array of bits. First of all, its size is independent of any discretization, giving access to much more precise simulations, and hence to much more precise results. Also, it provides much more satisfactory results from a technological point of view. The results from the array of bits representation are crippled by small holes, of the size of one grid square. Though this is demonstrating that the algorithm progresses toward the theoretical best solution (made of infinitely many holes of infinitely small size), it gives poor designs when it comes to actually build them. The Voronoi representation is a good example of how a smart yet explicit representation is able to avoid most of the problems of a more naive representation. However, its scalability remains a problem.

2.1.8 Optical fibers design

Manos et al. [81, 79, 80] introduce an explicit representation to design the cross-section of optical fibers. A cross-section of such fibers could be represented by symmetrical arrangement of non intersecting circles on a disc (see figure 2.6). Manos et al. devise a genotype which encodes the positions of circles in a disc quadrant, and the *desired* radii of the circles. When decoding the genotype, the circles are placed in the disc quadrant. Then, the circles radii, initially set to zero, are increased until they reach their desired radius, or until a circle intersection is about to occur. Finally, the quadrant is shrunk to an angular sector and repeated to fill the disc, thus creating a degree of symmetry specified by the genotype.

Thanks to the enforced symmetry, any genotype is valid, hence no constraint has to be taken into account during optimization. Huge numbers of circles do not imply a complex, long genotype, so the search space can still be well explored. Manos et al. used their representation with success to find competitive designs for various engineering problems related to optical fibers.

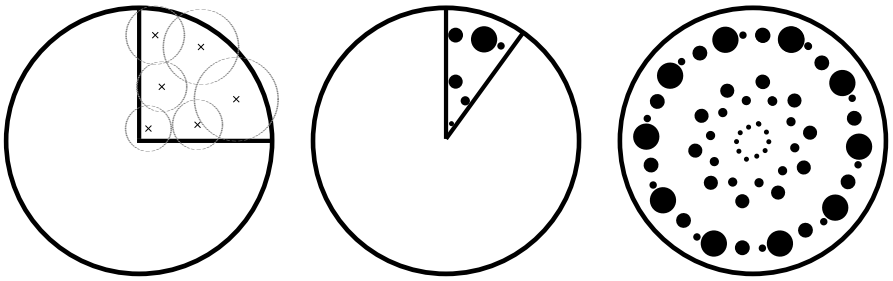


Figure 2.6: A representation for optic fiber cross-section, proposed by Manos et al. [80]. The genotype is a set of circles in a circle quadrant. The quadrant is shrunk, the circles radii are adjusted to meet the constraints, and the sector is then duplicated.

2.1.9 Blade shape design for wind turbines

Marc Ebner [25] and later Pavel Petrovic et al. [99], use Evolutionary Algorithms to optimize blade shapes for wind turbines. In both works, the candidate shapes are simulated within a rigid body simulator. The air stream is modelled as small spherical particles thrown at the the candidate shapes, and the rotation speed of the blade is measured over a time interval. This speed should be maximized.

Marc Ebner represents a shape as a tree of primitives, where the leaves represent a 3d shape (either a cube, a capsule or a sphere) and the internal nodes are affine transformations of groups of shapes (rotation and translation). Both leaves and internal nodes possess numerical parameters, to control respectively the shape sizes and the transformation applied to a group of shapes. The optimization is carried out by genetic programming, with its canonical setup as presented in [69]. Two results emerge from this work. First, when the internal nodes code for *relative* transformations, the mean best fitness of an optimization run is much superior than in the case of *absolute* transformations. Second, using a form of Lamarckism (*structure evolution*, introduced by Rechenberg [104]) do not improves the results over a pure genetic programming approach.

Petrovic et al. build upon the Marc Ebner work by adding two other representations for the same problem. A first representation is simply the coordinates of 26 points. Those 26 points are processed by a triangulation algorithm to build the candidate 3d shape. That representation is optimized with a real-valued genetic algorithm. Petrovic et al. also propose an alternative variation around the same idea: a tree with 26 leaves, the points, and internal nodes, that represents affine transformations of groups of points. The tree is interpreted to move the points and then the points are processed by a triangulation algorithm to build the candidate 3d shape. Both representations give better results than those obtained by Marc Ebner with his representation.

One criticism of those works is that the obtained solutions are not scalable: a

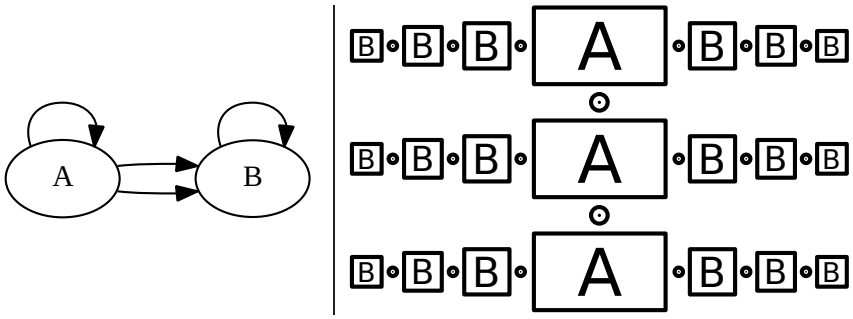


Figure 2.7: Genotype to phenotype mapping for the Karl Sims' virtual creatures [112], illustrated on a three segments, six legs creature.

solution is a blade for one kind of turbine, for one kind of flow. Furthermore, the information from the simulation are poorly used. For instance, some parts of a candidate blade might be hardly reachable by the small particles that simulate the air flow. Knowing the mean number of particles that hit each small square areas of a candidate shape might help to deform that shape into a better one. By representing a shape as an initial shape and a control policy that deforms that shape, by using a feedback from the simulation, scalable solutions might be possible. But this would no longer be an explicit representation.

2.2 Implicit representations

Implicit representations are those such as the phenotype size cannot be deduced with just the knowledge of the genotype size. Such representations are often using the genotype as an instruction sequence, a blueprint, to assemble parts, building a phenotype. Eventually, an external environment can influence such a building process.

2.2.1 Karl Sims' creatures

Amongst the works which demonstrated the role of Evolutionary Design as a "creativity lever", Karl Sims' virtual creatures [112] are considered a masterpiece. Sims' creatures are made of boxes, linked together by motorized mechanical joints. The joints are controlled by a neural network: each box could contain several neurons, and neurons from mechanically linked boxes are also linked. The neurons can have different activation functions, like oscillators or step functions. The neurons of a box could be connected to sensors, like light sensors, limb sensors, touch sensors. The goals are relatively simple tasks: ability to move forward as fast as possible, or the ability to follow a light. So the phenotype, here, was an assembly of boxes with its whole control architecture.

Sims encoded his creatures as an almost acyclic graph: a node can have multiple edges pointing to itself. The graph is transformed into a creature by a

fixed set of rules. A node describes a single box of the creature: its size, and its inner neural network. A graph link instantiates boxes and connects them to its source box with a limb. If a node has as recurrent edge, then the corresponding box will be replicated with eventually a geometric transformation (scaling, rotation, etc.). The replication can be done several times, this information is carried by a recurrent link of the graph. A non-recurrent link copies the result of the decoding of the destination node. This is exemplified in figure 2.7. Indeed, this graph encoding is very similar to an L-System encoding.

The variation operators created by Sims act both on the graph topology and parameters. Mutations of the node and edge parameters include boolean attributes flipping, zero-centered Gaussian perturbations of numerical attributes, etc. Mutations of the graph topology include edge source and destination node modification, addition and removal of edges and nodes. Sims also defines two crossover operators for graphs. The evolutionary algorithm is designed for a massively parallel computer: one candidate genotype is transformed to its phenotype and evaluated on one processor.

Sims' evolutionary runs end up to very different solutions. For the swimming, hoping and walking tasks, a wide variety of strategies are routinely found. Known mechanism design patterns, as counterweights, levers, flywheels emerge from the search process and are combined together in various fashions for the hopping and walking tasks. The swimming creatures can have lampreys like bodies and movements, or move by flapping large flat areas or have propeller-like structures. Combinations of those different mechanisms were common. The goal of Sims' work was to generate a wide variety of moving creatures, and in that sense it is a great success.

Thomas Miconi et al. [84] propose a simplification of the control architecture used by Sims, by using a single activation function for the neurons. They also introduce modifications in the way the neural network is built. Peter Krcah, in his master thesis [71], obtains significant improvements, in term of number of fitness evaluations, and solutions qualities. He replaces the rather canonical Evolutionary Algorithm used by Sims by an adaptation of the *NEAT* algorithm. Whereas the standard *NEAT* algorithm works on neural networks, as detailed in Appendix C, it is easily generalized to graphs with attributes, as those used by Sims for his creatures. Both Miconi et al. and Krcah obtained the same diversity of morphologies and behaviours as those found by Sims. That might be a sign that the design of creatures made of articulated boxes is indeed a relatively easy problem when using the Sim's graph representation.

2.2.2 Hornby's L-systems

Some of the most prominent achievements in Evolutionary Design where triggered by the L-systems, as in Gregory Hornby's work (Illustration of this work are shown in figure 2.9, 2.10 and 2.11). He devises mutation and crossover operators for parametric L-systems, working directly on a set of production rules. Those variation operators are able to add rules or to alter existing rules. While the genotypes are parametric L-systems, the phenotypes depend on the interpre-

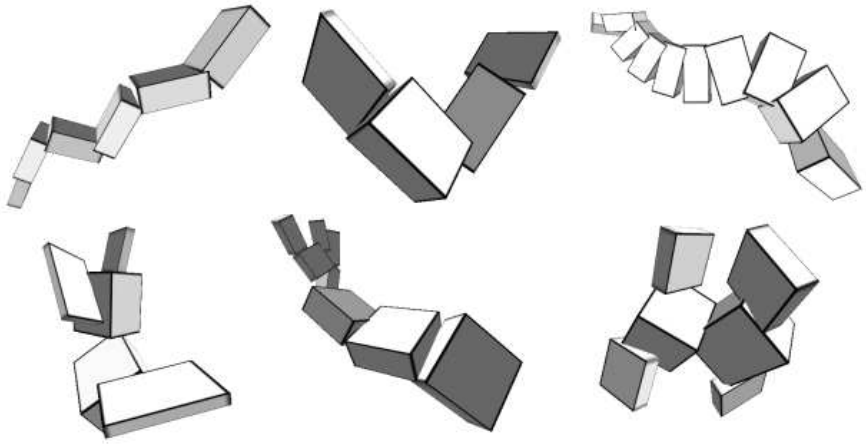


Figure 2.8: Some swimming creatures obtained by Peter Krcah [71], building upon Karl Sims' work.

tation of the symbol string obtained by computing the developmental sequence of an L-system. The problem of when to stop computing the developmental sequence is addressed by arbitrarily choosing a particular number of steps.

Hornby obtains 3D table designs [48], where the strings generated by the candidate L-systems are interpreted as moving instructions for a virtual point that creates matter in a discretized space. The objective function employed is the product of several simple objective functions, promoting hollow constructions, with a filled top and which are balanced.

The very same approach led Hornby to antennae design [77]: the strings generated by the candidate L-systems are interpreted as moving instructions for a virtual point. The obtained path (eventually a branching path) is then considered as a metallic wire in an electro-magnetic field. The quality of the wire as an antenna is deduced from the simulation and is used as fitness value. Amongst the best designs, one antenna was chosen to be part of the *ST5* space satellites. Those antennae are of better quality than what was obtained from more classical engineering techniques.

Finally, Hornby's approach has been used to explore robots design [51, 50]. The strings generated by the candidate L-systems are interpreted as moving instructions. Each move creates a segment (with eventually branching segments), and each segment pairs are either connected by a hinge joint, or a fixed joint. Each hinge joint is powered by an engine, driven by an oscillator. The obtained robots are evaluated within a simulator: the fitness is the distance between the robot position at time zero and after a few seconds of simulation. The best solutions found were built in real. Comparisons with direct approaches, or ADFs like solutions show the high superiority of the parametric D0L-system

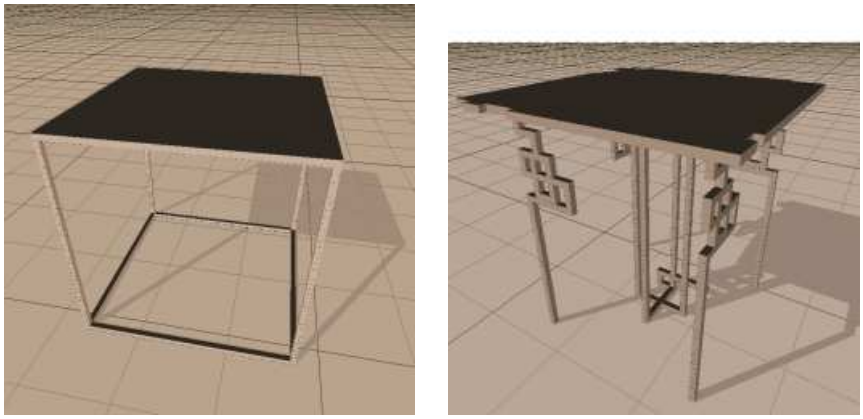


Figure 2.9: Examples of tables obtained by Hornby.

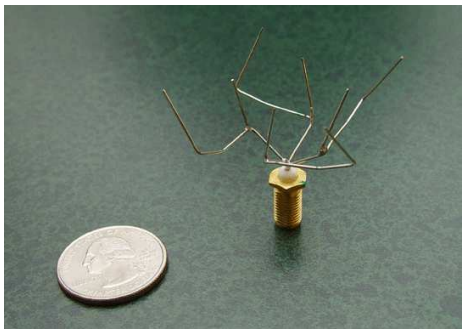


Figure 2.10: An example of the Hornby's antennae

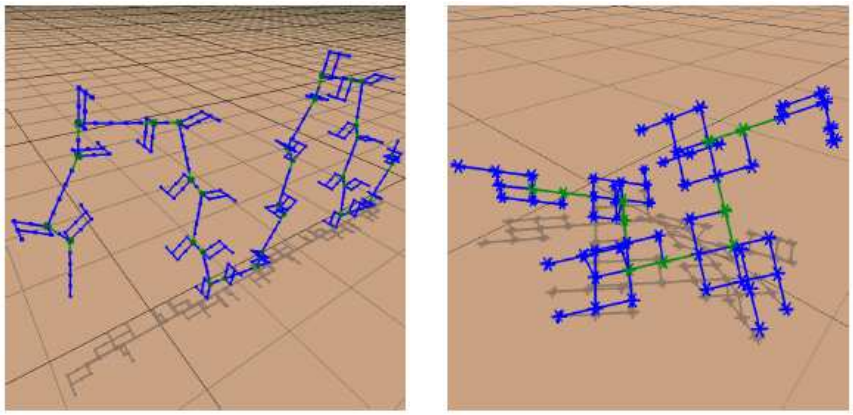


Figure 2.11: Examples of robots obtained by Hornby.

representation, both in term of quality and of fitness level.

In chapter 1 Section 1.3, we criticized the L-Systems based representations for their lack of *evolvability*. But Hornby successfully used such representations to optimize designs. Is this a contradiction? Our interpretation is that for the three problems introduced here, randomly chosen genotypes would have a high probability to perform well. First, in the case of the antennae design, as stated in the Section 2.1 of this chapter, current engineering methods are of little utility. Much simpler and explicit representations, as those introduced by Lohns et al. [76] can also beat traditional designs. Hornby's L-systems are coding for branching patterns, it might be interesting to see the proportion of *good enough* designs with this representation. We think that the same argument holds for table and robot designs. A reasonable proportion of robots, made of shapes linked together with engines and driven by oscillators, might be able of straight movement. No study has been made to check that proportion. If a high enough proportion of good designs were to be obtained, when randomly choosing genotypes, then evolvability of a representation would no longer be an important issue.

2.2.3 Environment aware L-systems

Environment feedback is possible with a parametric L-systems, as demonstrated in chapter 1, Section 1.3. This is what Martin Hemberg and Una May O'Reilly demonstrated with *Genr8*, for three dimensional surfaces design [43, 44, 45]. *Genr8* is intended to be an architectural design tool, to explore surfaces. The Hemberg et al. own extension of the parametric L-systems (named *Hemberg Extented Map L-Systems*) acts in the 3D space, generating segments that can merge, thus defining meshes. The generated segments are subject to physical forces: gravity, but also attractor, repellors, etc. Thus, the same L-system encodes different meshes when developed within different environments.

In order to carry out optimization of their own extension of the L-systems, Hemberg et al. use *Grammatical Evolution* [91]. With Grammatical Evolution, the genotypes are derivations from a given problem-specific grammar, coded as a string of integers, while the phenotypes are the result of this derivation. So Hemberg et al. devise a grammar that generates valid L-Systems rules, and genotypes are derivations of that grammar. The objective function is a linear combination of six objective functions. Changing weights of the combinations leads to different compromises between the objectives, thus to different shapes.

Hemberg's work demonstrates that a feedback from an external environment could influence, and then inform in some ways, the developmental sequence of an L-System. *Genr8* have been successfully used by designers and architects, as an *innovation machine*. From a pure optimization point of view, the very tenuous link between the genotype and the phenotype might be disturbing. The genotype is a string of integer, that is decoded into L-System rules, which in turn generate a shape. With such a mapping, the locality property (introduced in chapter 1, Section 1.1) is likely to be completely lacking, thus building a very hard optimization problem. But within the scope of *Genr8*, exploration of

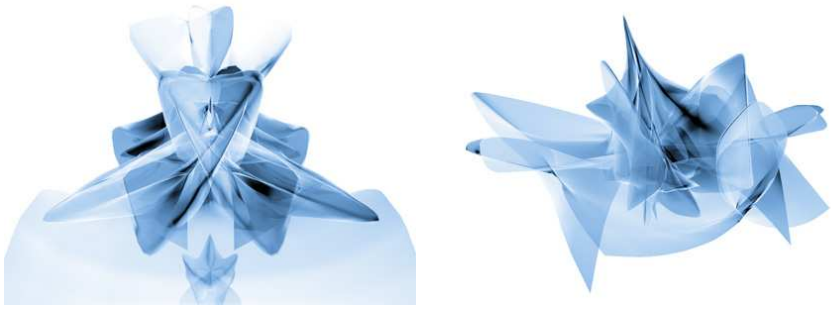


Figure 2.12: Examples of shapes obtained with the *Genr8* representation [43, 44, 45].

designs, this is not an issue.

2.3 Cellular representations

Computer graphics is a research field that also has to address representation issues. Many man-made or natural structures are very demanding in labor time to be modeled, and generative models could automate such tasks. As a consequence, Computer Graphics researchers designed a wide variety of structure generators. Indeed, L-systems are very common in computer graphics to generate trees and flowers models, as beautifully presented in [102]. In his PhD thesis, Kurt W. Fleischer [32] devises a generative approach to skin objects with organic looking highly detailed textures. His system can skin arbitrary meshes with fur, scales, spines, which have their own orientations and shapes, and form global skin patterns such as the snake rings or the leopard spots.

Fleischer's idea was to populate a mesh with cells. The cells are spherical entities that can split, migrate and exchange information through a simplified chemical model. The cells have an inner controller that takes the cells input (the level of some chemicals) and outputs decision: migration direction, triggering a split, etc... The inner controller is a simplified model of a biological gene regulatory network, built upon a system of differential equations. The cells chemical outputs are then interpreted to pick a color and a shape to build the mesh skin texture. The model is so sophisticated that it was proposed as a test-bed for synthetic biology.

Peter Eggenberger Hotz was among the pioneers to explore this idea for design. He first shows [26] that simple shapes could be the result of cell growth processes. His model of growth is simplified compared to the one developed by Fleischer, but retains many of its pattern generation capacities. Hotz uses his model of cell growth process to explore optimal lens shape designs [27].

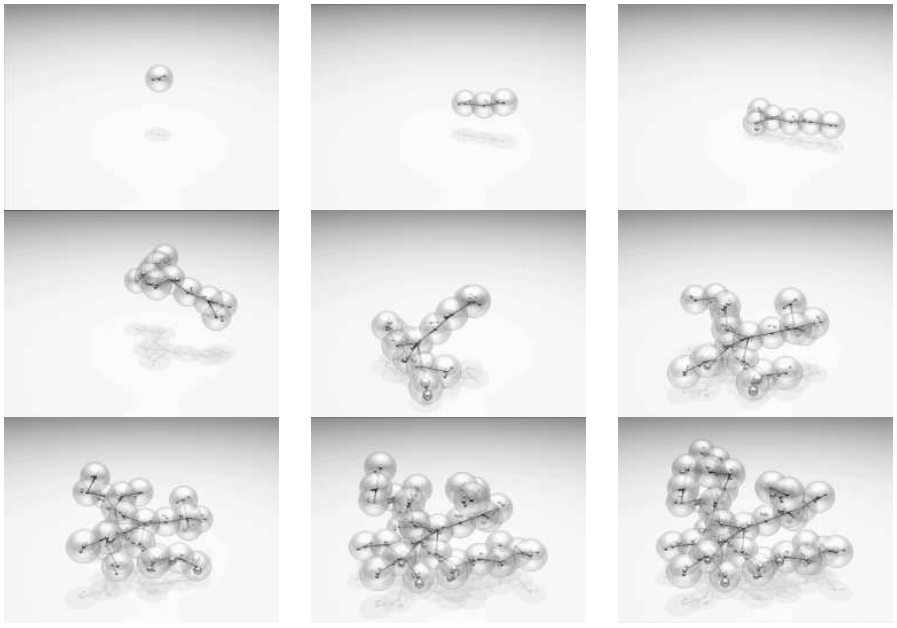


Figure 2.13: The developmental sequence of one of the Bongard's creatures [8, 10, 9].

2.3.1 Virtual creatures

Josh Bongard [8, 10, 9] uses a very similar model to the one developed by Peter Eggenberger Hotz, to generate virtual creatures, following in that way the work of Karl Sims. Bongard's creatures are assemblies made of a single base element, a cell. The creatures are encoded as a growth process, like Hornby's creatures. The growth is started from a single cell. A cell could be a cylinder or a sphere. The cells are connected together through a fixed quantity of slots, which act as motorized joints. Each cell contains a neural network, which takes environmental inputs (like touch sensors) and outputs the motor commands for the cell. Inside each cell, the decision of how to build the internal neural network and when to create new neighbours is done by a regulatory gene network, guiding the growth process. The creature regulatory gene network exchanges information through its cell slots. The growth process is evaluated for a fixed number of steps, and the grown creature is evaluated on simple tasks, like moving forward as efficiently as possible, or pushing blocks. The creature regulatory gene network is itself represented as a variable length bit string.

Bongard obtained creatures that were able to move in straight lines, search and push heavy blocks. These creatures featured emergent traits: some parts of the body play mostly a control role, while other parts play a locomotion role. The creatures are also quite modular (multiple occurrences of the same antenna or tentacle), despite the lack of any enforced hierarchy in the genotype. On the

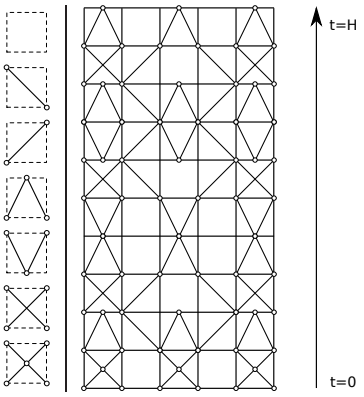


Figure 2.14: Kicinger et al. [63, 66, 64] represent a skyscraper frame as an array of tiles. A skyscraper frame is a truss, and the tiles are elementary trusses. Which tile type is used at a given location is decided by a cellular automaton.

other hand, no comparison is done with randomly generated genotypes, to see if the evolved creature traits are a strict consequence of the representation, or if the optimization also played a role in their emergence.

2.3.2 Skyscraper frames

Rafal Kicinger et al. [63, 66, 64] optimize the design of skyscraper frames. The frame of a skyscraper is a metallic truss, playing the role of the skeleton of the building. In Kicinger's work, a skyscraper frame is represented as a 2D array of size $W \times H$. The H rows define the roofs of the skyscraper, a row is made of W tiles. A tile can have one type from a total of 7 different types: the type determines the truss pattern to use for the tile. Figure 2.14 shows a 5×10 frame example and the different tile types.

Kicinger et al. use 1D discrete state cellular automata to decide the type of each tile of an array. A cell knows its own type and the type of it left and right neighbour. The row at height N corresponds to the cellular automaton state at the step N. The transition rules of the automaton are optimized with an ordinary bit string genetic algorithm, toward a structural quality criterion. Various settings of 1D, discrete state, cellular automata are used: totalistic or not, different number of cell states and transition rules, etc. Kicinger et al. compare their cellular approach with an explicit approach: a bit string that encodes the type of each tile of a skyscraper frame. Kicinger's results demonstrate the superior possibilities of the cellular approach. First, the solutions produced by the cellular approach are of better quality than those of the explicit approach. Furthermore, some of the best genotypes from the cellular approach are *scalable solutions*: they also express good solutions for larger or higher skyscrapers than the ones they have been evolved on. Finally, since a genotype of the cellular approach are the transitions rules of an automata, the size of the genotype is

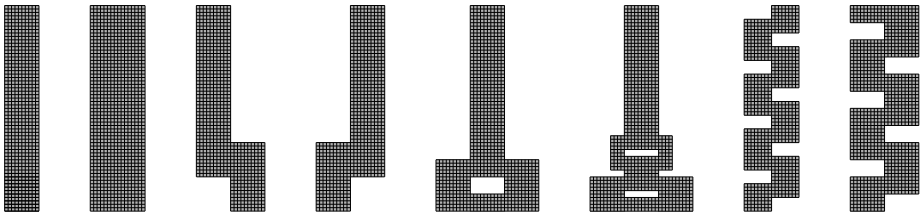


Figure 2.15: The different environments used by Kowaliw et al. to study truss growth processes represented with cellular automata.

independent from the size of the skyscraper. Thus, the transition rules are a *scalable representation* of a skyscraper frame.

Kicinger's work is original for at least two reasons. On the one hand, it is one of the first demonstrations of a cellular approach for a real world design problem: the cellular approach fulfills its promise of a scalable representation, providing scalable solutions. The second salient point is that the full developmental sequence of the cellular automaton is used: each step of the automaton codes for one row of the skyscraper frame. In all the other works we are reviewing here (and all those we are aware about), only the final state of the cellular automaton is used to define an object, not the succession of the states.

Due to the array and tiles environment, the skyscraper frames are very constrained, and the solutions presented by Kicinger et al. are not very original ones. An appeal of Evolutionary Design is the discovery of original solutions, so a more expressive representation would make sense. Another possible criticism would be the lack of environment-awareness. By completing the cell states with the local stress on the truss structure, one might obtain a better informed growth process, with potentially even better and more scalable solutions.

2.3.3 Truss structures

Kowaliw et al. use a cellular representation, Bluenome [68], to design 2D truss structures, and in some sense can be seen as a follow-up of Kicinger's work. The cellular representation is a discrete state cellular automaton on a square lattice. A cell state is a symbol from an alphabet ϕ of 33 symbols. The cells states are updated according to a set of rules, the same for all cells. A rule has a *predicate* part and an *action* part. The *predicate* part is made of 12 symbols from ϕ, while the *action* part states how a cell should be modified. For each cell, at each cell state update, the rule whose *predicate* best matches the 12 cell neighbour states is triggered. An action could be a change of the cell state, or a cloning of the cell to a nearby empty tile of the lattice. In addition, each cell has an age counter, which is decreased at each state update. When the age counter reaches 0, no more update is performed for this cell.

Initially, there is a single cell is. The cellular automaton is run for a fixed

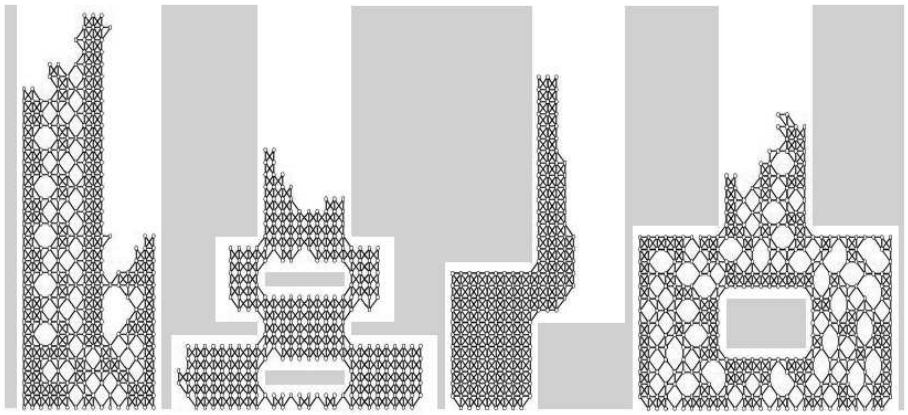


Figure 2.16: The same automaton coding for different trusses when run within different environments.

number of steps. The states of the cells are then interpreted by a hand-crafted, deterministic algorithm to build a metallic truss. Some tiles of the lattice could be marked as *fixed*, so the cellular automaton can not use those tiles as cells. The trusses are grown on constrained spaces: some tiles are fixed (see figure 2.15), where the cells cannot clone themselves. Thus, with this representation, a truss is encoded by a set of rules (the transition rules of an automaton), and the resulting truss depends on an environment. It should be noticed that arbitrary complex trusses could be grown with a same set of rules, it depends only on the environment size and the number of update steps of the cellular automaton. The representation is a *scalable* representation.

Kowaliw et al. represent a set of rules as a string of integers, optimized with a canonical genetic algorithm. The optimization objectives are

1. to fill the environment as much as possible with the generated truss.
2. to minimize the weight of the generated truss.
3. to have a structure as stable as possible.
4. to maintain the beam mechanical stress under an user-specified threshold.

Those objectives are aggregated into a single objective by multiplying them all together. After optimization, the best genotypes are able to express trusses much better than those obtained by random genotypes. Also, a good genotype is able to express good phenotype within various environments, even on environments different from the one used during the optimization process. Thus, Kowaliw's work, besides being a striking proof of concept of a cellular representation, also features environmental awareness. Although their representation is scalable, the solutions are scalable as well.

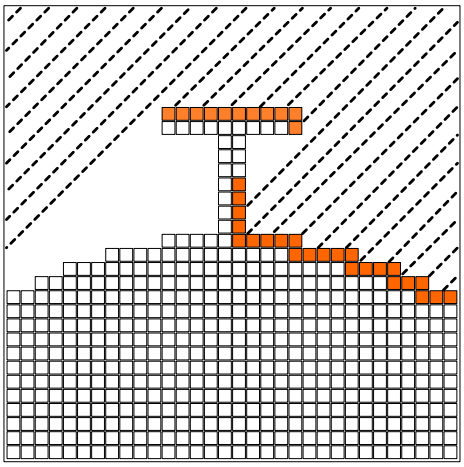


Figure 2.17: The environment used by Estevez et al. [30] to study environment-aware cellular representations. The environment is a square, regular lattice. Filled tiles (drawn as a square) are those which block the light rays (drawn as stippled lines). The light blocking tiles see their temperature increasing. The heat is diffusing through the environment.

Although Kowaliw's work is a clear and striking demonstration, it is not usable as is for real world truss structures optimization. Due to the underlying square lattice and the cell states interpretation process, only a subset of the truss structures is expressed. Trusses with different beams sections are not expressed, and the underlying lattice structure of the cellular automaton have a visible footprint, although a lesser one than in the Kicinger et al. work. Furthermore, the cellular automaton could be informed of the beam stress, to express better informed growth processes.

2.3.4 Environment awareness

We advocated the cellular representations, among other arguments, because they can be easily coupled with a relevant physical phenomenon. None of the previous examples features such a coupling, even if it seems appropriate and doable. Recently, Estevez et al. studied a cellular representation coupled with a heat diffusion simulation, first presented in [30] and highly detailed in [29]. Estevez et al. devise a simulated 2d environment, discretized as a 2d square grid. Each square can be either solid or transparent, and has a temperature. The solid squares block the light and absorb some energy, raising their temperature. The light comes as parallel rays. In addition, the temperature of the squares diffuses according to the heat equation. The problem is to turn some squares to solid state so that the average temperature of the whole grid is equal to a given, fixed level.

A possible explicit approach would be evolving a big binary string, one bit for

each square of the grid. Instead of that, Estevez et al. define a cellular automaton that takes the decision to add matter to a given square as a function of the nearby environment of this square. To evaluate the capacity of an automaton to build a shape that trigger the desired temperature profile, they update the environment grid according to the automaton rule for several steps, and then check the temperature profile. Between each environment update, the simulated physical state is updated, so each action of the automaton has a repercussion on the environment. What is optimized is the automaton rules, that defines a shape growth process. The cellular automaton definition is independent of the environment size and content, thus the representation is scalable.

Estevez et al. obtain umbrella-like shapes (see figure 2.18). Removing the feedback from environment, by shutting down the physics update, give inferior rules set. In the other hand, when the environment feedback is available, the optimized set of rules are able to generate good shapes in different environments. Furthermore, if some randomly selected filled tiles are set to empty, the cellular automaton is able to repair the umbrella-like shape.Thus, the best rules set are scalable answers to this temperature control problem. Estevez et al. show the feasibility of the idea to represent designs as the result of the interaction between a design building process, and the surrounding physical environment.

2.4 Conclusion

Evolutionary Design is driven by the desire to automatically design objects. The high diversity of objects designed by means of Evolutionary Algorithms shows at least how versatile those algorithms are. Decoupling the representation used for the stochastic search (the genotype) and the representation used for the evaluation (the phenotype) is the key of that versatility. When the complexity of the target design is bounded, an explicit approach can be highly successful. But most of those representations face a *scalability* problem. When using larger genotype in order to obtain more complex, hopefully better performing phenotypes, the Evolutionary Algorithm fails to trigger significant improvements. Investing more computing power becomes useless, as the search keeps enumerating uninteresting designs.

In order to solve that *representation scalability* issue, different kinds of representation have been introduced. Natural and human objects are often highly structured and regular, due to their *ontogeny*: the way they were built. One popular approach is to encode an object as the result of a building process rather than directly an object. Those representations are encompassed by the implicit representations. It begs, not a solution: how to represent a building process? An approach is to consider that objects can be defined as an explicit hierarchy of similar parts. Trees and direct acyclic graphs are a straightforward implementation of an explicit hierarchy. L-systems allows the designer to define the construction of a hierarchy. With a small set of rules, huge hierarchies can be generated, and hence, highly complex objects. However, though coupling a development process with a relevant physical phenomenon can be powerful, the

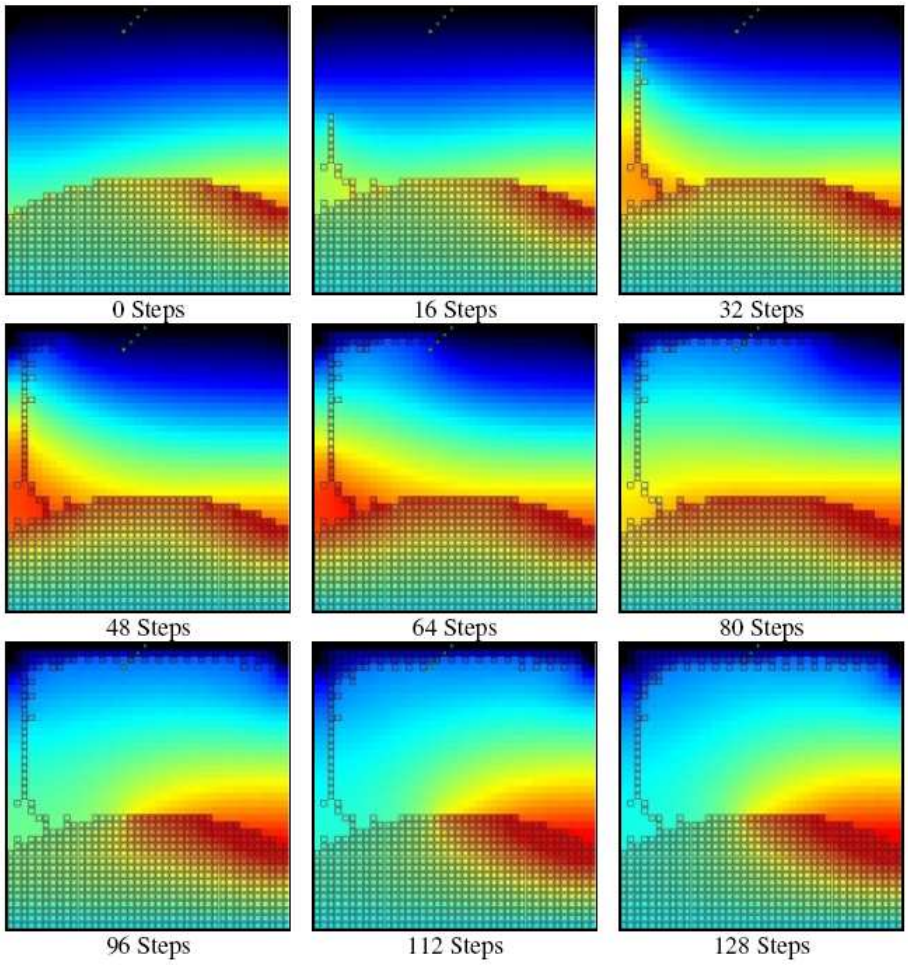


Figure 2.18: An example of development sequence with the Estevez et al. representation [30].

idea has not been much explored.

An important property for a genotypic representation is the *locality* property: small random variations of a candidate genotype should be likely to produce small random variations of its corresponding phenotype. We have demonstrated that an explicitly hierarchical approach can not fulfill such a property: a small variation near the top of a hierarchy produce huge changes. Similarly, two very close genotypes are likely to produce very different phenotypes. When the search space is scarce in good enough solutions, the optimization become intractable. So according to this reasoning, we find more promises in the representations of building processes that are non-hierarchical and fully decentralized.

Cellular representations are another implementation of a building process. One of its major feature is the total lack of explicit hierarchy, the building process is carried out by totally identical cells. Organisation, hierarchies, structures might appear, but emerge from the cells interaction. Because the behaviour of all cells is exactly the same, this encoding is very compact. Indeed the complexity of the genotype is independent from the size of the phenotype: this is a *scalable representation*. As with the L-systems, the building process can be coupled with a simulation. Evolutionary Design with cellular representations is quite recent, but the results are very encouraging. However, cellular representations come with difficulties on their own. The coupling with a simulation as been done only very recently, although the expected potential of coupling has been clearly demonstrated. The representation of a cell behaviour is still an open issue and no real consensus has yet emerged on this issue.

Another issue is the *control* of a building process. A *consistent building process* of an object should express an object with similar properties when triggered in similar situations. It might allow the reuse of a good building process on slightly different optimization problems, maybe at the cost of just a short run of the optimizer in order to fine tune the building process to the new problem. Such solutions to a design problem are *scalable solutions*. By evolving scalable solutions, we can solve multiple similar problems at a fraction of the cost of multiple full scale optimization runs. The idea of consistent building process is very close to John Rieffel's notion of good module [105]. However, even though this question is independent from the representation of a building process, it was not often addressed in the past indeed, being only present, as far as we are aware of, in the work by Estevez et al. [30].

Don't think about it. If you don't think about it, you won't sink. If you do, you will.

Mao Zedong

3

An Evolutionary algorithm for blueprints

The work described in this chapter aims at evolving complete structures from small atomic elements (such as Lego® bricks or Kapla® elements) in order to obtain walls, bridges and so on. Many representations have been proposed for such constructions (see chapter 2), but many of them easily lead to either actualizable structures (overlapping elements), or structures that are impossible to actually construct (even though no element overlap).

One way to overcome this difficulty is to indirectly represent a structure through a blueprint. Indeed, blueprints provide a rather expressive representation formalism, and Evolutionary Computation provides a way to evolve a plan (a genotype) such that the structure resulting from the application of this plan (a phenotype) is optimal for given objectives. One of the critical issues is then to provide evolution with efficient variation operators that explore some relevant part of the search space.

This chapter introduces *BlindBuilder*, a representation for indirect encoding of structures that uses a direct representation for blueprints, described as Directed Acyclic Graphs (DAG). The variation operators borrow from the developmental approaches of GP [39]. Experiments were conducted in order to analyze *BlindBuilder*, which lead us to what we believe to be an insightful discussion about the representations for Evolutionary Design paving the way toward cellular representations.

3.1 The BlindBuilder representation

Basically, a *BlindBuilder* individual is a directed Acyclic Graph (DAG) where nodes can be either *atomic elements* (the physical atomic elements of the construction) or *construction operators* (that define the way to fit some atomic elements together). No a priori assumption is given for the definition of operators and elements. It is hence possible to define a wide range of construction operators, as will be described later. More precisely :

- **Atomic elements** are terminals of the DAG (i.e. they don't have any argument since their arity is zero). However, they are not considered as

physical elements but rather as element templates that may be instantiated when needed. Each element template is defined with a given geometry and a physical behavior. Examples of atomic elements are Lego-elements, wooden shelves, tubes, wheels, artificial muscles, servomotors. Atomic elements are the base elements of the objects we want to evolve.

- **Construction operators** are functional nodes with a fixed number (*arity*) of arguments, i.e. targeted sub-nodes in the DAG, that specify what the defined function should be applied on (either other construction nodes or atomic elements). Numeric internal parameters parameterize the operator action (e.g. position for dropping a target element).

A well-formed *BlindBuilder* individual is hence a DAG such that the atomic elements are terminal nodes while the construction operators have as many sub-nodes as their arity. Moreover, there is a unique special node called the *top-level operator* (i.e. the entry point), so as to generate a single construction. When using such a DAG to build a structure, the construction program starts from the top-level operator, and iteratively builds the structure, evaluating each operators in turn until all terminal elements have been reached.

Thanks to this DAG-based representation, *BlindBuilder* is endowed with the combination of the following properties, which makes it a unique powerful representation formalism compared to those of the literature[1]:

1. *hierarchy*: the ability to consider as one single element what has already been built, as opposed to having to target specific sub-elements for any new operations. Once again, many previous works are about rather "flat" representations, without hiearchy, as introduced in chapter 2. An example of a hiearchical representation are the Hornby's L-systems [48].

2. *generality*: the representation can be easily extended to accept new kind of elements. This is a rare property amongs many previous works, as reviewed in chapter 2 are very specific. For instance, Petrovic [96] uses a representation specific to Lego plates, as well as Funes et al. who use a representation specific to Lego beams. In the other way, Peysakhov encoding [97, 98] features generality.

3. *3D representation*: some representations only consider 2-D structures, or don't scale-up well to 3-D structures. Here, no assumptions are made upon the geometric properties of the constructions to evolve.

In order to optimize candidate solutions expressed in this representation, we have defined a set of *variation operators* that build new solutions from older ones, and are described hereafter. In the literature, the optimization process performed in this chapter is referred to as graph-based Genetic Programming (i.e. population-based stochastic optimization using graphs, as detailed in [100], chapter 7 section 2).

[1] Note that in [49], some of these terms are used in the context of programs rather than graphs, with slightly different meanings.

3.1.1 Variation operators

Variation operators are used during evolution in order to generate offspring from parents. *BlindBuilder* only relies on mutation as variation operators – this is due to the great diversity between blueprints that makes it very difficult to perform meaningluf or even non-disruptive crossover. Two kinds of mutations are considered. The first kind alters only the parameters of a node in a blueprint, using a standard Gaussian mutation of standard deviation 10^{-1}. The second type of mutation are structural mutations, i.e. modification of the topology of the DAG, that can either add or remove a node[2]. In practice, there are five structural mutation operators, illustrated on figure 3.1:

1. *append*: A new non-terminal gets as input the top level node and, if its arity is greater than one, it gets other randomly chosen node as parameter. The internal numerical parameters of the new node are randomly generated using uniform distribution.

2. *replace* : the target node is replaced by a randomly chosen node with the same arity *and* its parameters are uniformly reset.

3. *fusion*: A node and all its input nodes are shrunk into one into a new operator with randomly generated parameters.

4. *reconnect*: The input edges of a node are randomly reconnected to other nodes then the operator parameters are renewed at random.

5. *permute*: The input edges of a node are randomly permuted and the parameters are randomly generated.

Any of the structural mutations are performed in such a way that the constraints on a blueprint (only one top level node) are never violated. If a mutation is impossible to apply, the plan remains unchanged. An individual is initialised using all the possible terminal nodes, and a single non-terminal mutation node is added, linked to randomly selected terminals nodes. It is important to note that these variation operators are *problem-independent* and can be used with any type of blueprint (i.e. any construction elements provided the corresponding construction operators are available).

3.2 Design of simple 3d objects

This section presents the experimental setup we used to test the ability of *BlindBuilder* to design simple constructions, based on two experiments. Each experiment relies on the use of a specific type of construction elements: stacks of bricks, which are 3d blocks with connectors on their surface, similar to the

[2]However concerned with variation of individuals between generations, the nature of our operators are inspired in part by the developmental approaches of [39]

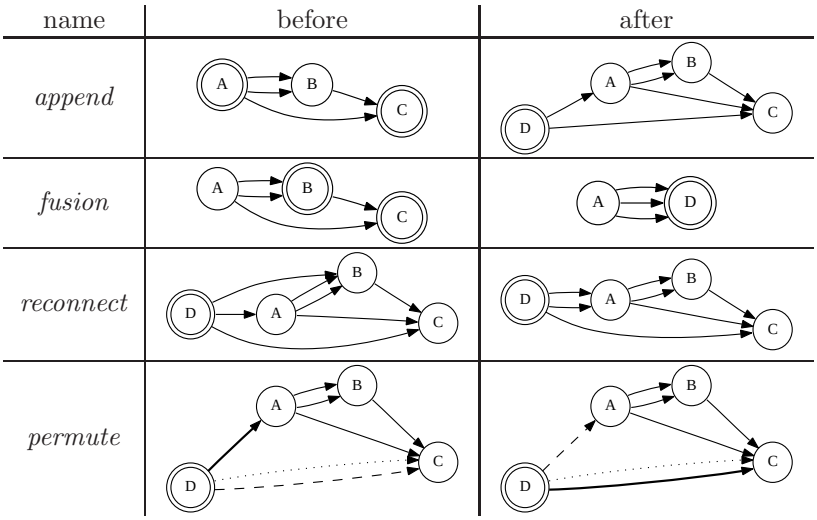


Figure 3.1: The mutations operators (the double circled nodes are the affected nodes)

Lego® bricks. The main difference with the Lego® bricks is that the connectors are defined only for positioning purpose, they do not exert any force.

A single *BlindBuilder* construction operator is defined, the SNAP operator, which builds a 3d brick stack by stacking together 2 other stacks. SNAP is formally written as : SNAP *[element1 target connector, element1 orientation connector, element2 target connector, element2 orientation connector] (element1 , element2)* . The *connector* arguments are used to pick up one actual connector from each argument-element (modulo the number of connectors of the actual argument), and the orientation of the connection is determined according to the *orientation* parameters (the number of parameters is thus independent of the size of bricks).

3.2.1 Experimental setup and hierarchical fitness

All individuals in the initial population are single-node DAGs, for which the unique terminal is uniformly chosen among the set of element templates. The selection is a tournament selection based on a hierarchical multi-criterion comparison operator. Two individuals are compared according to the the target objective(s), and to the size of their genotype (parsimony pressure). This comparison is made criterion by criterion, in a way similar to that proposed by [78]. Note that this is **not** a Pareto-based optimization (e.g., two individuals are always comparable).

- Define the relative distance between two values a and b as $\frac{|a-b|}{max(a,b)}$
- Order the list of objectives from most to less important

- Two individuals are said to be equivalent for a given objective if the relative distance between their values for this objective is less than a given threshold (typically 0.1)
- The comparison of two individuals is then lexicographic, i.e. individual x is better than individual y if, for some objective rank i, x and y are equivalent for objectives $1, \ldots, i-1$, they are not equivalent for objective i, and the value of x for objective i is larger than that of y.

In the following experiments, tournament size is set to 7 and population size to 1000. The threshold for the comparison of objective values is set to 0.1. All experiments have been run 13 to 20 times, stopped after 1000 generations. Each experiment took about 16 hours on a PC with Intel Pentium 4 running at 3.6 GHz under Linux. A few preliminary experiments (not shown here) showed that a Pareto approach (relying on NSGA-2 algorithm [21]) was slower than the hierarchical approach described above. Moreover, a standard generational GA (i.e. 1000 offspring are generated at each generation and replace all parents) using tournament selection was observed to be more efficient than both (μ, λ)-ES and $(\mu + \lambda)$-ES, with $\mu = 15$ or $\mu = 30$ and $\lambda = 7\mu$, the latter giving better result than the former (but see Section 3.3 for thorough parameter tuning). Finally, a maximal size of 50 for a blueprint was set to avoid uncontrolled code growth – but the limit was hardly ever reached.

The preliminary experiments also showed the relative importance of the variation operator *replace*: Indeed, this operator is much more conservative than the others, and is mandatory to fine tune existing structures, while all other operators result in important changes in the resulting structure. As a consequence, the rate for the *replace* operator is set to 0.7 while all other operators have a rate of 0.075 in the following experiments.

The *BlindBuilder* approach was implemented within *Open-BEAGLE*, a framework for artificial evolution written in $C++$ [35]. *Newton Game Dynamics*[3], was used in order to simulate and evaluate the resulting structure in a physical environment. All the experiments are in three dimensions.

3.2.2 The Pillar experiment

The goal is to build the biggest possible structure using brick elements (1×2,2×2 , 2×3, 2×4 and 2×6 bricks). The objective functions to maximise are, ordered by priority:

1. The *volume*: $V = \sum V_i$, where V_i is the volume of ith atomic element i.
2. The *compactness*: $\mathrm{C} = \frac{V}{V_{full}}$ where V_{full} is the volume of the convex hull of the whole structure.
3. The *parsimony*: $P = 50 - S$ where S is the number of nodes of the blueprint (max. 50 elements).

[3]Freeware but not open-source, see `http://www.newtondynamics.com/`

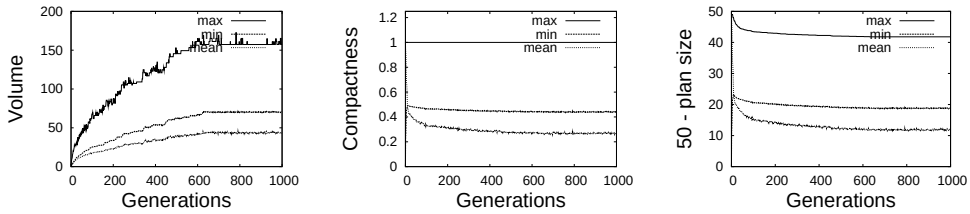


Figure 3.2: Maximum, average and minimum results for Pillar experiment using only the 2x2 Lego-element template. The different fitnesses are shown from left to right: volume, compactness, and parsimony.

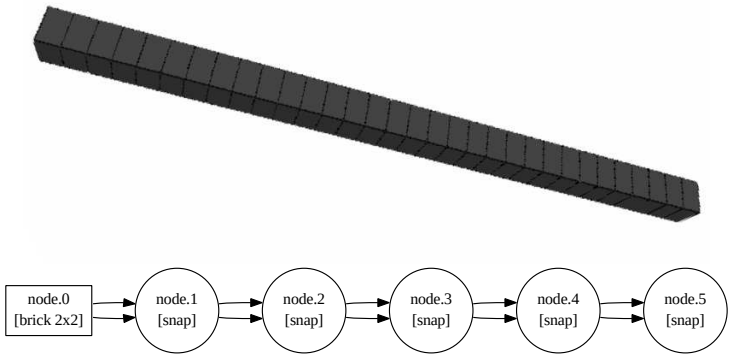


Figure 3.3: Example of best solution for the Pillar experiment using only the 2×2 Lego-element template. Snap parameters not shown.

Figures 3.2 and 3.3 show the evolution of all 3 fitnesses, and an example of one of the best structures, when using only the 2×2 brick. Results show that for this simple constrained problem, maximum compactness is achieved very quickly. Moreover, optimal individuals are found with the smallest possible blueprint. Figures 3.4 and 3.5 shows the same experiment when all the 5 brick sizes are used. The bigger and thus most appropriate element (2×6) is always used, even though the optimal plan is not yet reached at the end of evolution (it may be reached if evolution is carried on further). The two examples shown on figure 3.5 are very different blueprints, the latter being larger, but leading to a more compact construction. In all experiments, reusability has been heavily exploited, as can be observed in the sample plans of figure 3.5.

3.2.3 The Bridge experiment

The goal of this experiment is to build the longest horizontal structure with as few elements on the floor using a single brick type. The objective functions to maximise are:

1. The *length*, the horizontal length of the structure.

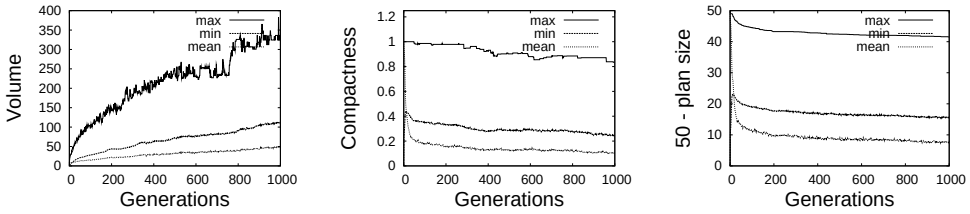


Figure 3.4: Maximum, average and minimum results for Pillar experiment using five possible Lego-element templates. The different fitnesses are shown from left to right: volume (with a different scale from that of figure 3.2), compactness, and parsimony.

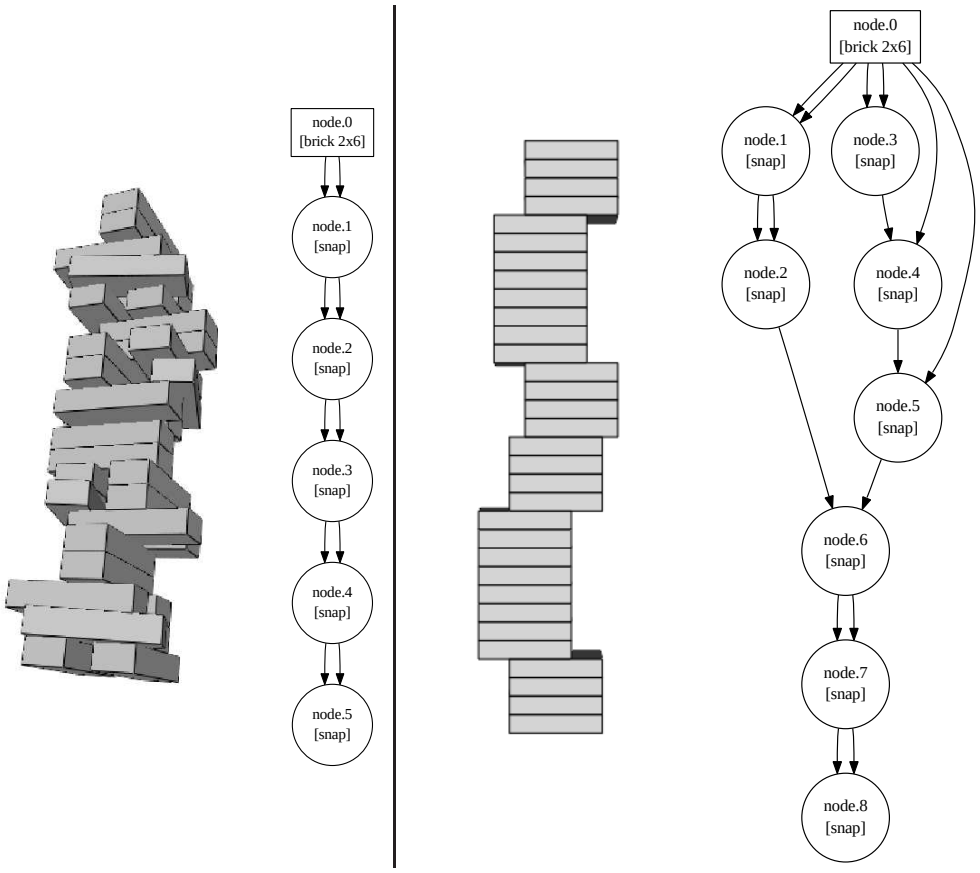


Figure 3.5: Examples of best solutions found for the Pillar experiment using five possible bricks type. Snap parameters not shown.

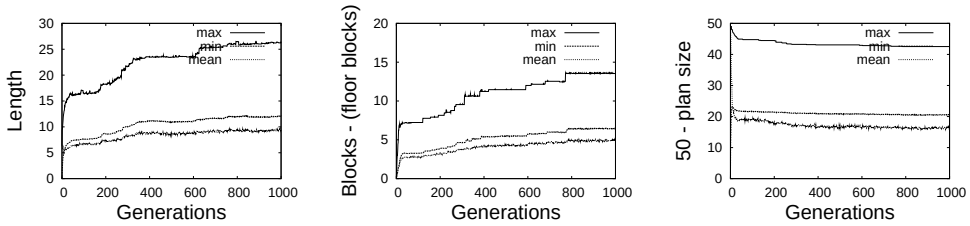


Figure 3.6: Results for the Bridge experiment

2. The *grounding*, $n - f$ where n is the number of atomic elements of the construction, and f the number of atomic elements that are in direct contact with the floor.

3. The *parsimony* defined as in the Pillar experiment above.

Figure 3.6 shows some resulting structures. Every run succeeded in generating quite successful individuals, either by deeply optimizing one of the objective functions, or by making a compromise between the three fitnesses. The most striking result is that evolution has been able to build *cantilever* bridges with *arches*, for which various examples are shown in figure 3.7. Each example represent the best individual for a given run. This figure also demonstrates the great variability between runs.

3.3 An empirical analysis

This section presents an experimental setup in order to find the best evolutionary setup for *BlindBuilder* on two simple construction problems involving more sophisticated construction operators than in previous Section. Then, on the basis of those experiments, we discuss the possibilities offered by *BlindBuilder* as well as its limitations, and propose some improvement directions.

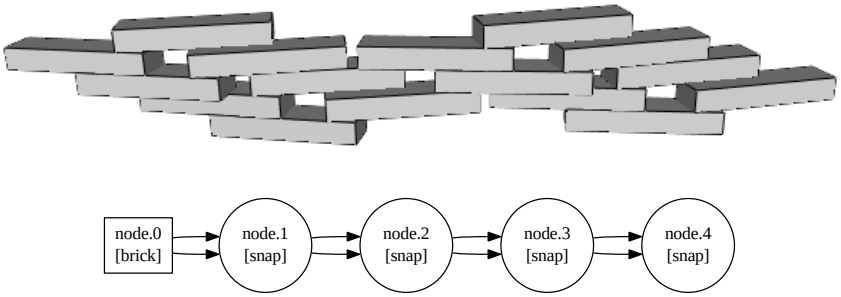
The framework for the constructions considered here is that of 2D stacks of identical blocks, similar to the block stacks introduced in chapter 1, Section 1.3.1. We provide four different construction operators that take 2D stacks as inputs, and output one 2D stack, and are illustrated by figure 3.8.

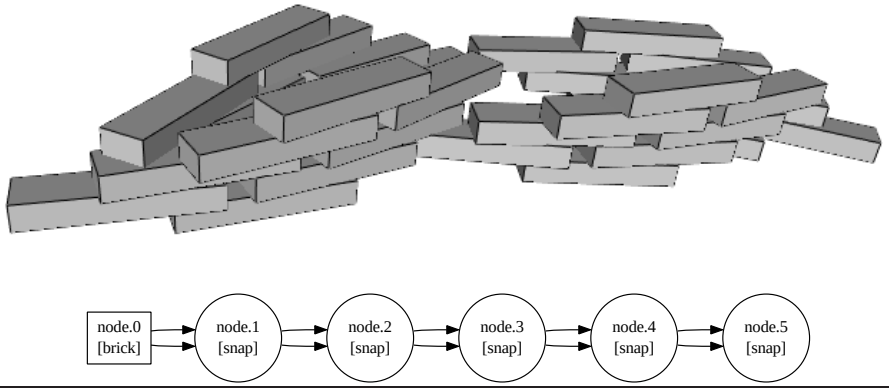
1. *drop* puts a stack of blocks on another group, with a variable shift. This shift is specified as a fraction of the length of a third stack.

2. *flip* flips a whole stack upside-down (symmetry w.r.t. an horizontal line)

3. *mirror* also flips a stack, but horizontaly (symmetry w.r.t. a vertical line)

4. *remove_top* removes the top layer of blocks from a stack.





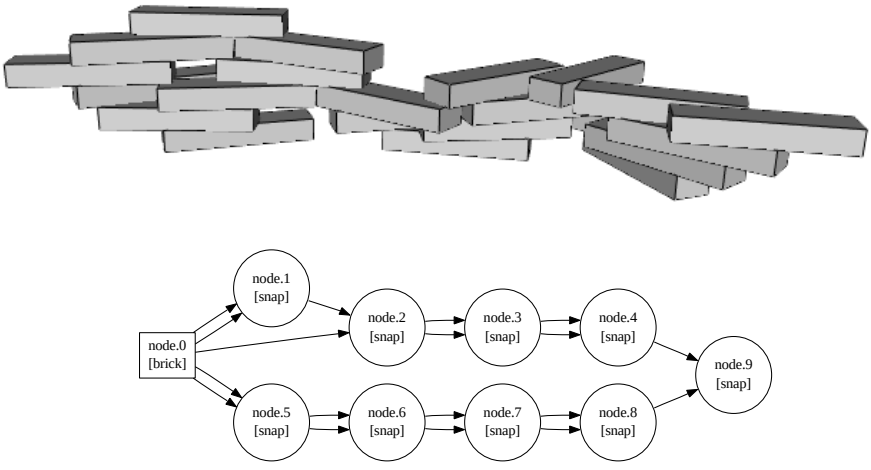


Figure 3.7: Examples of best solutions. Snap parameters not shown.

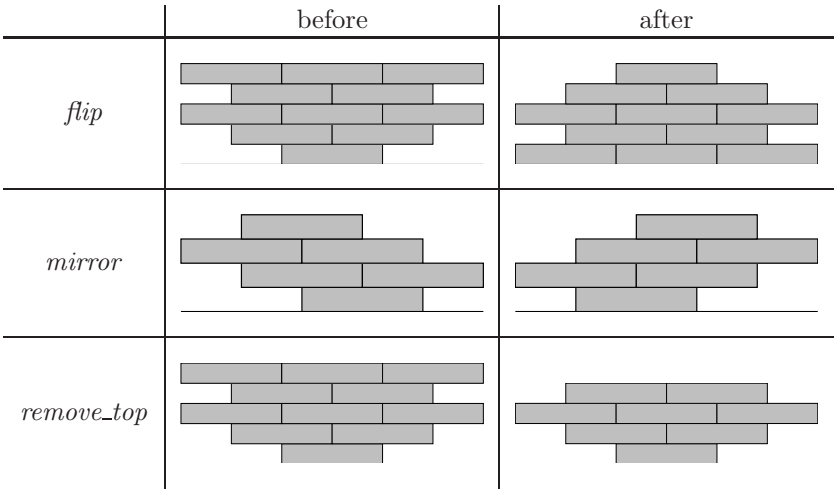


Figure 3.8: Action of the building operators.

3.3.1 Experimental settings

All experiments have been performed on two evolutionary design problems, the tower problem and the bridge problem, that are presented here, together with the corresponding objective functions. Then, the evaluation process that makes it possible to compute the fitness of an individual, i.e. the adequacy of a given candidate solution with regard to the objective function, is described.

The tower construction problem

The tower problem is simply defined as a maximization problem where the objective is to maximize the height of the construction, namely:

$$\frac{min(H_t, h)}{H_t}. \tag{3.1}$$

where h is the height of the current building h, and H_t is the desired height of the tower (user-specified). In all experiments here, H_t is set to 300 times the thickness of all blocks.

The bridge construction problem

The bridge problem consists in building a bridge defined as the longest construction including a "roadway" above some artificial 'water level', having as few blocks as possible in contact with the ground. For this problem, two parameters must be taken into account: the height of the water H_w, and the length of the bridge L_b. Let h be the height of the construction, $length(x)$ the (horizontal) length of block x, and $\Phi(height)$ be the set of blocks that intersect

the horizontal line at a given *height*. The goal is then to maximize the following three objectives:

1. Minimal height ($fitness_1$):

$$\frac{min(H_w, h)}{H_w} \tag{3.2}$$

2. Roadway coverage ratio ($fitness_2$):

$$\frac{1}{L_b}\left(\sum_{x\in\Phi(H_w+1)} length(x)\right) \tag{3.3}$$

3. Shore coverage ratio ($fitness_3$):

$$1-\frac{1}{L_b}\left(\sum_{x\in\Phi(0)} length(x)\right) \tag{3.4}$$

As in Section 3.2, the comparison of two individuals is a lexicographic comparison involving the 3 fitnesses above, in this order. In the following, the *threshold* is set to 0.01, H_w is set to 4 times the thickness of all blocks, and L_b to 14 times the length of a block.

Evaluation Process

As shown in figure 3.9, candidate solutions are evaluated in a two-steps process: "building" and "evaluation". Firstly, blueprint are interpreted in order to build the corresponding structure. This step is very fast to compute. Secondly, the structure is evaluated within a solid body physics simulator[4] during 250 simulation steps of 0.1 seconds of simulation time. This second step is a constructibility test, computationally expensive. Furthermore, during this second step, some stacks may be out of equilibrium and fall down, leading to an arbitrary very bad fitness for the individual at hand. If the structure is balanced with regard to the simulation, the 3 fitnesses described in previous Section are computed. The combination of "smart" construction operators and a costly but accurate constructibility test ensures the constructibility of the resulting structure while maintaining the computational cost at a reasonnable level, because only a subspace of the possible constructions is explored.

3.3.2 Experimental Protocol

Design Of Experiment

In next Section, we provide results of a factorial Design of Experiments for the two objective functions described above. Table 3.1 displays the parameters

[4]We used Newton Game Dynamics - see `http://www.newtondynamics.com/`.

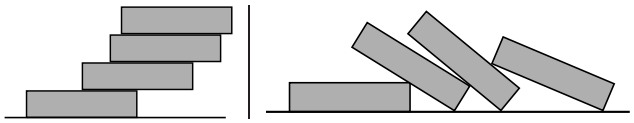


Figure 3.9: Building and evaluation processes: before evaluation (left) ; after evaluation (right).

Evolution Engine	$(\mu + \lambda)$-ES	(μ, λ)-ES
population size	1, 10, 50, 100, 250, 500	
λ/μ ratio	1, 2, 3, 5	
parameters mutation prob m_p	0.15, 0.35, 0.55, 0.75, 0.95	

Evolution Engine	generational with tournament
population size	10, 50, 200, 500, 1000
tournament size	2, 5, 8, 10
reproduction prob	0.3, 0.5
parameters mutation prob m_p	0.07, 0.21, 0.35, 0.49, 0.63

Table 3.1: The 440 settings experimented

we experimented with, and the corresponding set of values used in the DOE. Note that only m_p, the mutation probability of the parameters, and r, the reproduction probability in the generational case, are varied. The probabilities m_s of each the $N_p = 4$ structural mutation operators are the same, and is computed as the following: $m_s = \frac{1-(m_p+r)}{N_p}$. A run is stopped after 10000 evaluations.

Comparison Protocol

The results of two parameter settings are compared as follows: 20 runs are performed for each setting. Every 5 evaluations, the distribution of the best fitness in the population are compared using a Student T-test with 95% confidence. A parameter setting is considered better than another one if it is better for at least 90% of the evaluation steps, else we call it a draw. In order to rank all parameter settings, each setting is compared to every other setting, getting 1, 0 or -1 point if it is better, similar or worse respectively. After all possible comparisons have been made, the settings are ranked according to their cumulated scores.

3.3.3 Results

Figures 3.10 and 3.11 show the results for both the tower and bridge problems.

The Tower construction problem: The best settings are those with a tiny population and with the lowest probability of parameter mutation (i.e. high

Rank	Evolution engine		Pop size	Mutations settings		Score
1	(μ, λ)	$\lambda/\mu = 2$	1	mut param = 0.15		439
2	tournament	size = 2	10	mut param = 0.07	repro = 0.3	434
3	tournament	size = 5	10	mut param = 0.07	repro = 0.5	431
4	tournament	size = 8	10	mut param = 0.07	repro = 0.3	431
5	tournament	size = 10	10	mut param = 0.07	repro = 0.5	430
6	tournament	size = 8	50	mut param = 0.07	repro = 0.5	427
7	tournament	size = 5	10	mut param = 0.07	repro = 0.3	421
8	tournament	size = 10	50	mut param = 0.07	repro = 0.5	420
9	(μ, λ)	$\lambda/\mu = 1$	1	mut param = 0.15		418
10	tournament	size = 10	50	mut param = 0.07	repro = 0.3	417
11	tournament	size = 5	50	mut param = 0.21	repro = 0.5	417
12	tournament	size = 8	10	mut param = 0.07	repro = 0.5	415
13	tournament	size = 2	10	mut param = 0.07	repro = 0.5	410
14	tournament	size = 10	50	mut param = 0.21	repro = 0.3	409
15	tournament	size = 10	10	mut param = 0.07	repro = 0.3	406
16	tournament	size = 10	10	mut param = 0.21	repro = 0.5	406
...						
431	$(\mu + \lambda)$	$\lambda/\mu = 5$	100	mut param = 0.95		-397
432	$(\mu + \lambda)$	$\lambda/\mu = 3$	250	mut param = 0.95		-397
433	$(\mu + \lambda)$	$\lambda/\mu = 5$	500	mut param = 0.95		-398
434	(μ, λ)	$\lambda/\mu = 5$	250	mut param = 0.75		-399
435	(μ, λ)	$\lambda/\mu = 1$	500	mut param = 0.95		-410
436	(μ, λ)	$\lambda/\mu = 1$	50	mut param = 0.95		-412
437	(μ, λ)	$\lambda/\mu = 1$	250	mut param = 0.95		-415
438	(μ, λ)	$\lambda/\mu = 1$	100	mut param = 0.95		-419
439	$(\mu + \lambda)$	$\lambda/\mu = 3$	500	mut param = 0.95		-423
440	$(\mu + \lambda)$	$\lambda/\mu = 5$	250	mut param = 0.95		-423

Table 3.2: The best and the worst settings for the Tower fitness case (440 settings)

structural mutation rate). Most of these settings use tournament selection, except for the noteworthy exception of number 1 and 9 (μ, λ). Similarly, the worst settings use large populations, a high probability of parameter mutation and either (μ, λ) or $(\mu + \lambda)$ evolution engines. Note that all experiments converge towards the highest possible tower, composed of piled up blocks (see figure 3.10) : only the convergence speeds differ. For this problem, the *BlindBuilder* representation is very efficient and the result of setting number 9 highlights the fact that success is more related to representation issues (including construction operators) rather than optimization parameters. Indeed, setting number 9 corresponds to a random walk.

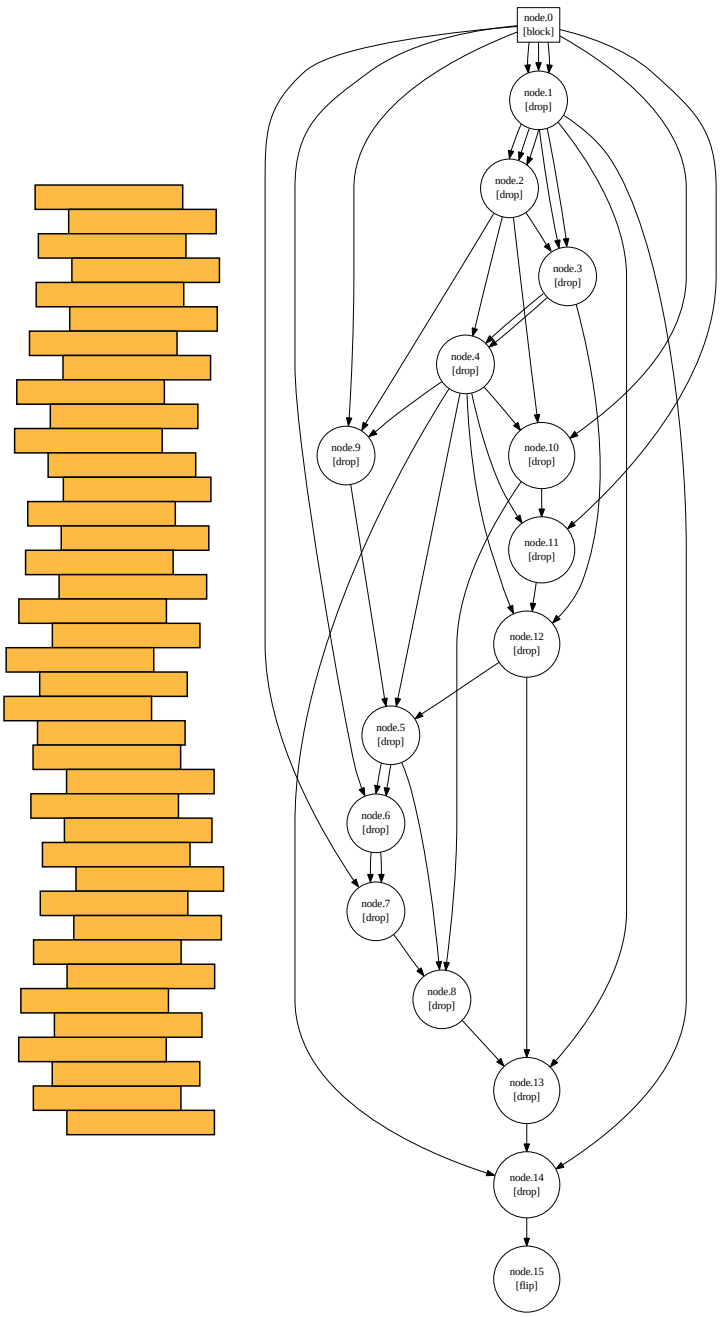


Figure 3.10: A tower structure and its corresponding plan

Rank	Evolution engine		Pop size	Mutations settings	Score
1	tournament	size = 8	10	mut param = 0.07 repro = 0.5	159
2	(μ, λ)	$\lambda/\mu = 3$	1	mut param = 0.15	151
3	(μ, λ)	$\lambda/\mu = 2$	1	mut param = 0.15	151
4	tournament	size = 10	10	mut param = 0.07 repro = 0.5	151
5	tournament	size = 8	50	mut param = 0.07 repro = 0.5	151
6	tournament	size = 2	10	mut param = 0.07 repro = 0.3	150
7	tournament	size = 10	50	mut param = 0.07 repro = 0.5	146
8	tournament	size = 5	10	mut param = 0.07 repro = 0.3	145
9	(μ, λ)	$\lambda/\mu = 5$	1	mut param = 0.15	139
10	(μ, λ)	$\lambda/\mu = 1$	1	mut param = 0.15	139
11	tournament	size = 5	10	mut param = 0.07 repro = 0.5	139
12	(μ, λ)	$\lambda/\mu = 1$	1	mut param = 0.35	135
13	tournament	size = 5	50	mut param = 0.07 repro = 0.3	131
14	tournament	size = 10	10	mut param = 0.07 repro = 0.3	128
15	tournament	size = 10	50	mut param = 0.07 repro = 0.3	128
16	tournament	size = 8	10	mut param = 0.07 repro = 0.3	127
...					
151	(μ, λ)	$\lambda/\mu = 3$	50	mut param = 0.95	-137
152	(μ, λ)	$\lambda/\mu = 1$	50	mut param = 0.95	-139
153	(μ, λ)	$\lambda/\mu = 5$	1	mut param = 0.95	-143
154	(μ, λ)	$\lambda/\mu = 2$	100	mut param = 0.95	-145
155	(μ, λ)	$\lambda/\mu = 2$	50	mut param = 0.95	-147
156	(μ, λ)	$\lambda/\mu = 2$	10	mut param = 0.95	-148
157	(μ, λ)	$\lambda/\mu = 5$	100	mut param = 0.95	-149
158	(μ, λ)	$\lambda/\mu = 1$	100	mut param = 0.95	-150
159	(μ, λ)	$\lambda/\mu = 3$	10	mut param = 0.95	-153
160	(μ, λ)	$\lambda/\mu = 5$	10	mut param = 0.95	-158

Table 3.3: The best and the worst settings for the Bridge fitness case (160 settings)

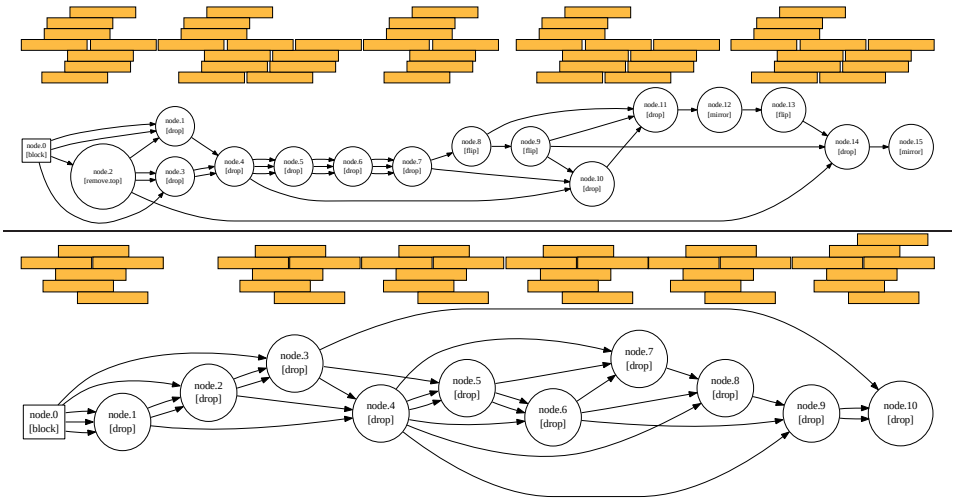


Figure 3.11: Some of the bridges structures obtained with their corresponding plans

The Bridge construction problem: Compared to the tower problem, very different bridge structures have been evolved (see figure 3.11), although they share similar properties: underwater pillars and over-water pillars that guaranty stability of the roadway are frequently encountered. Here again, high structural mutation rate along with tournament selection get the best results, but the successful population sizes are larger than for the tower problem: in this case, simple random walk performs poorly, and the population-based approach seems more efficient. Moreover, high selection pressure for tournament selection are preferred, and reproduction rate does not seem to have any significant impact. As for computational cost, the best runs took on average 2 hours on a single 2 Ghz CPU.

For validation purpose, we also performed an experimental campaign using classical canonical GP with ADFs – replacing DAGs with trees. GP terminal set corresponds to *BlindBuilder* terminal nodes with operator parameters (depending on the context), and GP function set corresponds to *BlindBuilder* operators. It should be noted that there is a mapping between *BlindBuilder* blueprints and tree-based GP from the final construction viewpoint. Results (not shown here) were very disappointing, even though we tried common GP settings: tournament selection of size 8, population size of 100 and 500 individuals, either 90% subtree mutation (no crossover) or 90% subtree crossover (no mutation), and several ADFs number and depth. Runs were much slower, for the same number of evaluations (3 to 10 hours versus 1 to 3 hours with *BlindBuilder*) due to the well-known bloat phenomenon: best individuals were mostly made of unused code. One possible reason for that massive bloat is the inadequacy of the standard GP operators : subtree mutation or crossover lead most of the time to infeasible candidates. We also conducted limited DOE runs, in order to try to

improve the first results and find better parameter settings, but results (and the CPU time requirements) were discouraging enough, even for the tower problem, so that we did not continue any further. From all our experiments, tree-based GP with ADF is not adapted for the Evolutionary Design problems considered here.

To conclude with our approach, we observe that as the problem becomes more difficult (bridge vs. tower), evolution parameters increase in relevance (vs. stochastic hill-climbing): larger populations perform better, while stochastic selection/replacement (tournament selection) always performing better than the other evolution engines we tried. In summary, we could say that exploration (via structural mutation and tournament selection) is favored over exploitation (parameter mutation, more conservative approach during selection/replacement with ES-like evolution engine). However, when the size of the constructions increases, we begin to pay the price of such an exploration strategy, as fine tuning of the operators parameters become necessary, but is difficult. For example, the evolved tower constructions are more and more unstable during the course of evolution, mostly because the blocks were not precisely aligned. Note that this could be addressed with a Baldwinian or a Lamarckian approach, adding some local fine tuning of the construction parameter at evaluation time (e.g. on-line optimisation). Nevertheless, this would not suppress the problem of poor performance for mutation on parameters.

3.4 Discussion

The experiments presented in this chapter produced results comparable or even bigger in size than that of the literature, with (sometimes dramatically) less evaluations (Funes et al. [34, 33], Petrovic [96], Peysakhov et al. [97, 98]). However, it quickly becomes very difficult to expand existing structures even though both construction and variation operators could make it possible. This is caused by the fact that even a small modification on a construction operator is likely to be more and more disruptive as the size of the graph increases (i.e. altering deep construction operators can be very disruptive). Indeed, the less disruptive way to expand an existing individual is to add new construction operators to the blueprint, rather than to modify existing ones. As a consequence, trajectories in the search space differ greatly between different runs, even with the same parameter settings[5].

To address this issue, we could experiment with Islands Model [2] in order to better exploit hardware resources by eliminating non-promising runs. Anyway, we believe that it could only hide a fundamental problem without really solving it, that is the problem of scalability, only partly addressed using Island Models: the major results of our work is that current representation for indirect encoding are not truly scalable. This conclusion has lead us to consider two possible extensions of graph-based representation for Evolutionary Design. The

[5] When only the seeds for random number generator differ.

first is based on adding recursion, and the second is to consider context-aware blueprints. We shall describe these in turn.

In its current form, *BlindBuilder* representation is already able to reuse substructures in the construction process. However, this could be made much easier by adding construction operators defined as a parameterized production rule. For example, such a production rule could recursively build a subgraph with only the number of recursive calls as parameter (e.g. $produce(A, 5)$ would create the subgraph $A \leftarrow A \leftarrow A \leftarrow A \leftarrow A$, with A a unary construction operator). Once defined, such production rules are interesting because of the very few parameters to deal with (i.e. the number of recursive calls) - moreover, resulting blueprint should be shorter. However, the introduction of production rules would also give to *BlindBuilder* an unwanted feature of the L-systems. With such rules, two blueprints that differs form each other by one single mutation are likely to express very different things, thus performing very differently. Between the compactness asset and the disruptivity flaw, we are not sure of the real effectiveness of such a sophistication.

Another approach is to take advantage of the environment, i.e. to consider context-aware blueprints. In the standard setup, a blueprint is interpreted in an environment so as to produce the candidate structure. In some case however, it is possible to rely on information *from* the environment during the construction process: for example in the bridge problem, the water level is considered to evaluate the resulting construction, but could also be used to trigger construction phases. This would have many advantages: optimizing blueprints that could be executed in different environments (i.e. different water levels, different length needed) as well as keeping blueprints short (i.e. simple recursive process to build the underwater part of the bridge independently of the water level). Instead of using the information provided by simulations only for fitness evaluation, we could also use it to tune the parameters of the construction operators. For instance, for the bridge and the tower problems, knowing the forces balance on each block would allow us to define a more insightful placement of the blocks. The difficulty of growing blueprints would be at least reduced. Compared to adding parameterized construction rules, the emphasis would then be put on the on-line construction process rather than off-line optimization. Of course, both approaches can easily be combined, since they tend to address the same issue (scalability) along with being compatible (addressing smaller construction operator and developmental process triggering).

3.5 Conclusion

In this chapter we have addressed the problem of Evolutionary Design with *BlindBuilder* a new representation formalism which is hiearchic, and more general with regard to other works in the literature. We have focused on specific construction and variation operators for optimizing classical Evolutionary Design problems. We conducted an extensive set of experiments in order to provide an in-depth analysis with comparable representations in the field of Evolution-

ary Design.

Our conclusions have lead us to reconsider the current approach, and to highlight the drawbacks of commonly known constructive encoding approaches. In particular, the optimization process tends to be more and more inefficient as the size of graphs increases. While the specific properties of *BlindBuilder* make it possible to reduce the impact of such drawbacks, it cannot really scale up. Nevertheless, our in-depth analysis of the construction and optimization process have led us to identify the current conceptual roadlocks and to draw some insightful conclusion and perspectives on representations for Evolutionary Design.

We have in particular highlighted several perspectives for our work to address before-mentioned problems. The first approach, yet more classical, is to expand our representation formalism with new construction operator embedding production rules. The second approach, more original, intends to take into account the very environment so as to perform a context-aware construction process, enabling both greater adaptivity to environmental variations and smaller blueprints. Indeed, by using cellular representations, a more promising approach than production rules based approaches can be setup, while being able to exploit environment awareness. Thus, the further chapters are focused on cellular representations.

One Ring to rule them all,
One Ring to find them,
One Ring to bring them all,
And in the darkness bind them.

The Lord of the Rings, J. R. R. Tolkien.

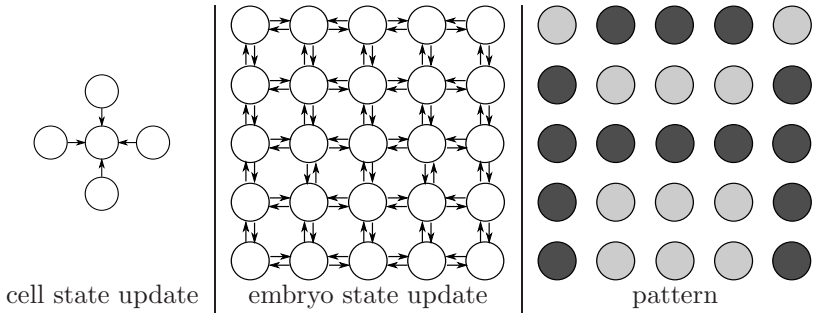
4

Embryogenic flags

Evolutionary Design needs representations that are able to express modularity while demonstrating good evolvability and being easily coupled with a physics simulator. Potential candidates, in this quest for a perfect representation, might be cellular representations. Those representations describe an object as the result of repeated interactions of *cells* within an environment. The cells are usually controlled by some *controller*, that only has a local view of its environment. While initially the cells might be in the same state, later, because of interactions, their states, and thus their actions, will become different from each other. At some point in time, the global state of the system is the phenotype of the representation, and is evaluated with respect to the task at hand. The genotype, on the other hand, is the common controller of all cells. Cellular representations can thus be encoded easily, e.g. as plain strings of symbols or vector of real values. In particular, the size of the genotype is completely independent from that of the phenotype. Furthermore, because of their local connections and their local action, such cellular systems are easy to tightly couple with any numerical simulation that also uses (and computes) local information, like a Finite Element Analysis. Finally, once a good control is found, the same control might also be fitted to slightly different environmental conditions, or to similar conditions at larger scales.

As reviewed in Chapter 2, recent experimental results promise a lot. However, there is at the moment no clear consensus on how to setup a cellular representation. What kind of controller is suitable for the cells? How cells should diffuse information to yield a global coherent behavior? How to ensure that a cellular representation express itself into similar designs within slightly different environment?

The *de facto* testbed to study cellular representation is the *embryogenic flag*. After an informal introduction of the embryogenic flag problem in Section 4.1, we introduce a more abstracted view of the embryogenic flags, discussing different theoretical solutions, from the trivial ones to the more complex ones in Section 4.2, before surveying the related literature in Section 4.3. We then present in detail our experiments on cellular development processes. Different controller and optimizer associations are studied according to the same protocol, described in Section 4.4. Section 4.5 details the use of general recurrent neural networks op-



timized with the *NEAT* evolutionary algorithm. Section 4.6 shows our usage of *Echo State Networks* with the *CMA-ES* Evolution Strategy. Section 4.7 introduces a new cellular control model reminiscent of the reaction-diffusion systems, also relying on *CMA-ES* for the optimization part. Furthermore, Section 4.8 underlines the strong importance of the stopping criterion of the developmental phase, illustrated by a simple experimental setup.

4.1 Introduction to embryogenic flags

Embryogenic flags[1] are two dimensional cellular automata, where each cell is one of the tiles of a two dimensional square grid. A cell has its own individual state. The cells states are modified by an *update function*, which computes a new cell state from the states of its neighbours and its own state. Additionally, each cell computes an output value, artificially transformed into a colour. This colour could be either part of the cell state, or computed from the cell state by an *expression function*. All cell colors, on the 2D grid, thus form a graphical pattern, and the goal is to design a cell controller so that the pattern matches some given target pattern.

No one definition is necessary. It can be either discrete or continuous values, either scalar or vector values. Also, while each cell has its own individual state, all cells share the same update function. Moreover, throughout this work, we will consider that initially, all cells are in the same state. An embryogenic flag is hence completely defined by its update function and this shared initial state: an embryogenic flag is a potentially compact representation of a picture. In particular, the size of the representation is independent from that of the picture. On the other hand, the picture encoded in a given embryogenic flag can only be unveiled by repeatedly applying the update function to all cells, all cells being initialized to their common initial state: this process is called the *development* of the embryogenic flag. From an Evolutionary perspective, the development is the decoding function, that computes the phenotype from the genotype.

[1]The words *embryogenic flag* will be abusively used for both the cellular representation (the cellular automata discussed in this section), and the problem of finding the controller of the cells of a cellular automaton so that it generates the target picture.

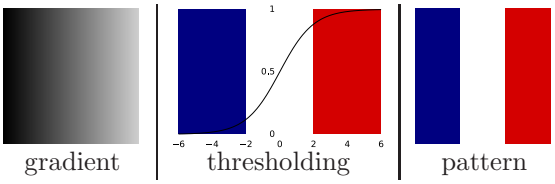


Figure 4.1: From a concentration gradient of a morphogen, a three band pattern can be built. The decision of which color to pick is performed by applying thresholds on the local concentration of morphogen.

During its development phase, an embryogenic flag can also be viewed as a Dynamical System [62]. In particular, the long-term behavior of an embryogenic flag during its development can be one of the following: It can reach a *fixed-point* (sometimes called *steady state*), a global state that is its own successor; It can have a periodic behavior, cycling through the same global states; or it can have a totally chaotic behavior (only if the space of possible global states is infinite). Moreover, some steady states can be *attractor* states: if the system is in such a state and is perturbed by some (small) noise, further development steps will restore the same steady state. Such a property is often termed *self-healing*, and an important issue is then the size of the basin of attraction of the steady state, i.e. the amount of noise that can be added while preserving the return to the same steady-state.

The challenge of embryogenic flags is to be able to find an update function such that a given target picture can be reached at some point of the development phase, starting from a given (uniform) initial state. This problem is reminiscent of the *French flag model* proposed by Lewis Wolpert in [121]. In the French flag model, the French flag is used to show how biological cell differentiation could be carried out just by local mechanisms. In this model, cells emit some chemicals, called *morphogens*, that diffuse through space. Additionally, the cells are sensitive to the local concentrations of morphogens. As the system evolves through time, the cells can build different concentration maps of the morphogens. By associating two thresholds with the local concentration of morphogens, cells can ultimately 'decide' if they are either red, or blue, or white, forming a pattern somewhat similar to, say, a French flag (see an illustration on figure 4.1). Although here, the motivations are not at all the modeling of biological phenomenons, a very similar issue is to be addressed: global pattern formation through local interactions.

The *embryogenic flag* problem raises issues that are similar to those raised in the area of *amorphous computing* [1]. Amorphous computing refers to computational systems that use large numbers of identical parallel processors. Each processor has limited computational ability and only local interactions. The processors are considered to be faulty, asynchronous and distributed over space. As in the case of the embryogenic flags, the local states and interactions can lead to a global coherent state. Various algorithms and strategies were proposed to maintain gradients, lines and other simple patterns and shapes, despite the

purely local, and faulty, interactions. R. Nagpal wrote a nice and short introduction [90], while J. Werfel provides a much more extensive view [119]. So far, in amorphous computing, the whole system is crafted by hand, without involving any automatic optimization. Also, the feasibility of the approach with real hardware is an important issue in amorphous computing. Moreover, one of the main concerns in amorphous computing is to find formalisms and languages to ease the engineering of such distributed and decentralized systems.

However, in the context of Evolutionary Design, the long term purpose behind the study of embryogenic flags is to develop our ability to construct objects that are the result of interactions between cells in a given *simulated* environment, and hardware implementation is not an issue. On the other hand, the simulation of the cellular system is embedded in an evolutionary loop, meaning that the computational algorithmic efficiency is a relevant issue. Also, we are concerned about some forms of *scalability*: the ability to carry out the same computations at various scales (implicit modularity) and in various conditions (environment awareness). Additionally, the desired self-healing property can be viewed as a corollary of scalability: a self-healing development process should exhibits the same functionality in spite of noise, and under varying conditions, thus meeting John Rieffel's definition of what is a good module (see Section 1.3, Chapter 1 our discussion about modularity). On this aspect, amorphous computing provides interesting answers. For instance, L. Clement [15] proposed a general way (actively maintained gradients) to build a segment pattern by using only local interaction between cells. If the cells states are perturbed by some external noise, the segment is rebuilt. Such a constructive and robust segment definition is what we believe to be a consistent, scalable definition of a segment in the framework of Evolutionary Design, too.

4.2 An insight of the embryogenic flag problem

In this section, we discuss the *embryogenic flags* from a more formal perspective.

An embryogenic flag is a two-dimensional regular square lattice of *cells*, with finite width and height. Each cell has it own state, can read its neighbours states (and its own), and can output its state to its neighbors. The neighbourhood of a cell can be, for instance, the Moore neighbourhood (8 neighbours) or the Von Neumann neighbourhood (4 neighbours) shown in figure 4.2.

An *update function*, the same for all the cells, computes the state of a cell from its inputs (neighbours states, plus eventually its own). The update function might be applied either *synchronously* (all cells are updated in the same time) or *asynchronously* (the cells are updated one after the other, following a given order). Asynchronous update can be done either randomly, or according to a user-defined order (e.g. updating the cells rows by rows). The updates are repeated until some *stopping criterion* is met. The stopping criterion acts based on the global state of the system.

An *expression function* takes as input the state of a cell and outputs a symbol from an alphabet, or a real value. In the present work, the output of

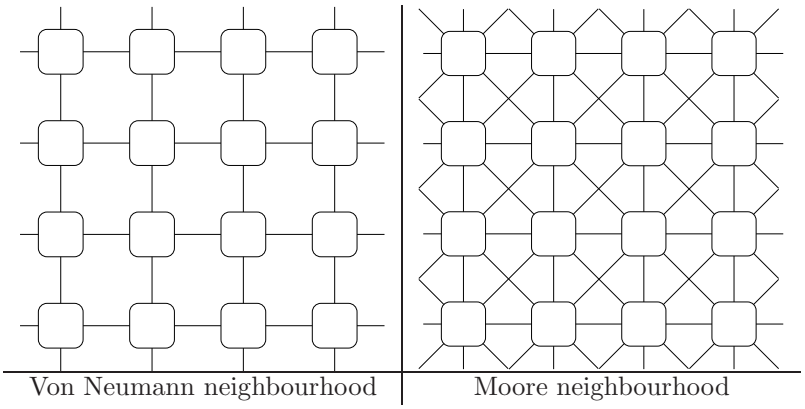


Figure 4.2: 4×4 embryogenic flags with differents neighbourhood

the *expression function* will be a continuous value in $[0, 1]$, easily visualised as a grayscale level. Applying the expression function to the whole flag generates a two-dimensional pattern. The *expressed pattern* of an embryogenic flag is the pattern computed by the expression function after the stopping criterion has been met. The expression function and the update function are often unified, called the *controller* of a cell.

The embryogenic flag problem is an optimization problem: find an update function such that, from a given initial state of the embryogenic flag, the expressed pattern matches as closely as possible a given target pattern. The only information available to guide the search (the objective function of the optimization problem – the fitness function in case of evolutionary optimization) is a measure of similarity between the expressed pattern and the target pattern.

Secondary goals of the embryogenic flag problem are to find *self-repairing* and/or *self-healing* solutions. After the stopping criterion has been satisfied, a random noise is applied to all cell states, and development is resumed. If the embryogenic flag can restore its global state to the one it had before the perturbation (or a one that is very close to it), the flag is *stable*. Of course, strong perturbations might break permanently the expressed pattern, while smaller perturbations might just temporarily disturb the pattern: the *stability* of the flag can be quantified: different kinds of perturbation should be studied, local or global, bounded (e.g. uniform) or unbounded (e.g. Gaussian), ...

4.2.1 Direct Solutions

An embryogenic flag is fully determined by its cells initial states and its update functions. However, if the initial states of the cells match the colours of the target pattern, a trivial solution to the problem is to choose for both the update and the expression functions the identity function. Furthermore, if the initial states of all cells can be different, and are part of the genotype of the embryogenic representation, solving the problem simply amounts to solve a regression

problem. Unfortunately, in such a case, the size of the genotype is the size of the target pattern: the representation is not a compact representation, there is no hope for scalability or self-healing properties. Moreover, it is clearly desirable that the initial state does not depend on the target pattern.

4.2.2 Non-Developmental Solutions

Even with this restriction, there can exist trivial non-developmental solutions to the embryogenic flag problem: an initial state made of two orthogonal homogeneous linear gradients can do the trick (as any other strictly increasing or decreasing function). Let us consider an $n \times m$ embryogenic flag, and initialize cell (i, j) ($1 \leq i \leq m$ and $1 \leq j \leq n$) with state $(u_{i,j}, v_{i,j})$ defined by

$$(u_{i,j}, v_{i,j}) = \left(\frac{2i-1}{2m}, \frac{2j-1}{2n} \right)$$

and choose the identity function as update function. Solving the embryogenic flag amounts to solve a simple regression problem for the expression function, using the maximization of the similarity of the expressed pattern with the target pattern as objective. For instance, a multi-layer perceptron can be trained to approximate the target pattern as closely as wanted (by adding more neurons in the hidden layer), using either backpropagation or an Evolution Strategy, depending on the nature of the similarity measure used as objective function.

Since no development takes place in this case, we call this solution the *non-developmental solution*. Note that this solution is in some aspects scalable: for a given precision, a larger target pattern does not require more neurons in the hidden layer of the Multi-Layer Perceptron. However, the perceptron would need more neurons for a more complex pattern. Indeed, the same phenomenon would appear for any other function representations, each one having a good or a bad bias depending on the target pattern. Also, as for the direct solution above, self-repair is here impossible. If the states are perturbed, the cells have no ways to restore themselves. In this non-developmental solution, the spatial organisation uses some global knowledge, whereas in the Evolutionary Design context, we want to use as much as possible only local knowledge.

4.2.3 A generic developmental solution

From the non-developmental solution, a simple and effective developmental solution could be built. Instead of building the two orthogonal homogeneous gradients in a single step, it could be done in an iterative way, mimicking a developmental process, starting from an initial configuration where all cell states have the same value. In Appendix A, we introduce the *Springy Comb* system, a one dimensional continuous cellular automaton that exhibits as unique global attractor a one-dimensional linear gradient. It is easy from the *Springy Comb* to build a 2D embryogenic flag where each row and each column implements a *Springy Comb*, choosing the right inputs and combining them carefully. This generic development solution has a perfect self-healing property: Whatever the

noise introduced in the states, they will return to the linear gradients after some development iterations.

4.2.4 Reaction-diffusion systems

Reaction-diffusion systems were introduced by Alan Turing in his seminal article [116], as a model of pattern formation from an homogeneous, uniform state. Those systems are partial differential equations systems, whose general form is:

$$\frac{\partial q}{\partial t} = D \triangle q + R(q)$$

A reaction-diffusion system describes how chemical concentrations, the q vector, evolve over space and time. The chemicals diffuse isotropically in space, each with its own particular speed. The diffusion speed of each chemical is specified by the diagonal diffusion matrix D. The reaction function R specifies how the different concentrations of chemicals increase or decrease over time, depending on the concentrations q of other chemicals.

For instance, in [116], Turing studied a reaction-diffusion system with two chemical species, a and b. His system is defined as follow:

$$\begin{aligned} \frac{\partial a}{\partial t} &= s(k(16 - ab) + d_a \triangle a \\ \frac{\partial b}{\partial t} &= s(k(ab - b - 12 + \beta) + d_b \triangle b \end{aligned}$$

Fixing the diffusion rates such as $d_b = \frac{1}{4} d_a$, and $\beta = 12$, the system will always converge to a steady state where a and b forms spots. Other β values will give steady states where a and b are arranged as stripes. This system's steady-states are illustrated on Figure 4.3.

Reaction-diffusion systems are known for the wide variety of patterns they can express. Changing the reaction function and the diffusion coefficients governs the type of resulting patterns, as well as their stability. Variations around spots and stripes patterns are the best known [117]. Karl Sims [111], by interactively evolving the reaction function and the diffusion rates, was able to generate stripes, spots, spirals waves, fractals, and combination of those. Sanderson et al. [109] show how to combine different reaction-diffusion systems to build new ones that produce hybrid patterns. They also show how to use an anisotropic diffusion term to add geometric warping to the patterns. If the chemical concentrations are perturbed, the system will return to a different steady state, but those steady states will always share a set of properties: stripes patterns, spots patterns, ... If the steady state of a system is a stripe pattern, then the system will always rebuild stripes once the perturbations are lowered to a small enough level. Such a reaction-diffusion systems are robust and scalable definition of stripes.

A reaction-diffusion system might or might not reach a steady state: it depends on the initial conditions, the reaction function and the diffusion coefficients. For instance, the system proposed by Turing can have a chaotic behavior

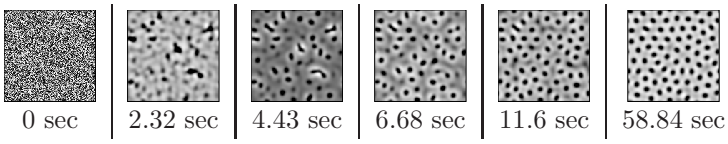


Figure 4.3: Simulation of a reaction-diffusion system (128×128 grid), using the 2 chemicals reaction proposed by Alan Turing. Time information shown is the simulated time. The second chemical is displayed as a gray level. At steady state, the flag is filled by spots of the same size and with the same gaps. They appear if the chemicals are initially taken from a random uniform distribution.

if the parameter β is lower than 8. Although predicting the steady states of a reaction-diffusion can require a lot of analysis or can be even intractable, detecting a steady state is rather easy. If the variations of the chemicals are null, the system will not change further. Hence a stopping criterion might be:

$$\left\| \frac{\partial q}{\partial t} \right\|^2 \leq \varepsilon$$

Reaction-diffusion systems are defined over continuous time and space, but are usually discretized both in time and space. If we consider a reaction-diffusion system over a square plate, discretized into small squares, and an explicit Euler integration scheme, the reaction-diffusion system can be rewritten as follows:

$$q(i,j)_{t+1} = q(i,j)_t + \Delta_t D\text{laplacian}(q_t, i, j) \ + R(q_t)$$

where the Laplacian operator is approximated by

$$\text{laplacian}(f,i,j) = f(i-1,j) + f(i,j-1) - 4f(i,j) + f(i+1,j) + f(i,j+1)$$

With a sufficiently small time step Δ_t, and by forcing the chemicals concentrations q to be above 0, some of the most well-known reaction functions can be (slowly) approximated[2]. Indeed, this discretization can be seen as the update function for an embryogenic flag, with a Von Neumann neighbourhood. Such update function is an example of a totalistic update function: it considers only the sum of the neighbouring chemicals. The expression function can then be defined as a trivial function that simply returns a given chemical level, say the first one.

Also, reaction-diffusion development exhibits an interesting form of self-healing: after some perturbation, a system might not come back to exactly the same global state, but will stabilize into different steady states that share a set of common traits. All the work is done in the update function, whereas the expression is reduced to one chemical concentration. Using a universal function approximator (like a Multi-Layer Perceptron or a Radial Basis Function

[2]Implicit integrators are a much better choice (unconditional stability and better accuracy), but this not the point here

network) as a reaction function, we can optimize its weights and the diffusion coefficients in order to approximate any pattern. Though, the difficulty of such an optimization problem remains unknown.

4.2.5 Towards non-trivial developmental solutions

While the generic developmental solution built upon gradient generation can approximate any pattern in theory, it has some flaws. If the target pattern fits well to a Cartesian coordinate system (like horizontal stripes or a checker pattern), the regression of the expression function will be easy. But if the target pattern features radial symmetry (like a disc or a star), the expression function will have to encode the transformation from a Cartesian to a radial coordinate system. If fact, hoping that such a transformation will be discovered without any clues out of the pattern similarity measure is very optimistic. Depending on the representation of the expression function, such a transformation might be expensive in term of genotype size, and thus in optimization time. On the other hand, if the cells are able to express a self-repairing radial gradient, radial patterns would be obviously simple and Cartesian patterns quite challenging.

This is the central challenge of the embryogenic flag problem. On the one hand, the cells can generate self-repairing patterns. While any pattern could in principle be expressed, a guaranteed self-healing is easier to achieve on the simplest pattern (linear gradients for instance) but might be harder on complex patterns. On the other hand, the expression function can interpret a simple gradient to build complex patterns, at the expense of its complexity.

Discretized reaction-diffusion can express complex patterns, and all the work is done in the update function, but which pattern can be expressed is unknown, as well as how to guarantee the self-repairing. Hence a delicate balance has to be found between the complexities of update function and the expression function. An ideal update function should be able to express a rich diversity of simple self-repairing gradients. In addition, small changes of the parameters of that ideal update function should lead to small changes of the expressed gradient. An ideal expression function would be simply a universal function approximator. Multi-layer perceptron or radial basis functions networks are natural candidates in this category.

4.3 Survey of the Embryogenic Flags

All studies on the embryogenic flag problem focus on how to find good update functions for developmental solutions. However, as of today, no real consensus has been reached, and the different works on the embryogenic flag problem are quite difficult to compare. We will nevertheless try in this section to discuss different aspects of existing works, from the model used for the update function to the neighborhoods connecting the cells, the update order, the initialization, the possible birth or death of cells, the stopping criterion of the developmental process and the emerging self-healing properties of the resulting controllers.

A discussion about the relevance of the different approaches surveyed here to Evolutionary Design will conclude the Section.

Also tables 4.1 and 4.2 give a quick comparative overview of all works mentioned here along some of the different axes listed above.

4.3.1 Update Function Models

In all works surveyed here, the proposed update functions are partly ad-hoc (user-defined, and fixed along evolution), and partly optimized by an evolutionary algorithm. Figure 4.4 is a symbolic representation of the range of possible cases.

The interest on the embryogenic flag problem was largely initiated by the seminal work of Julian Miller and his co-authors [85, 86]. Their works provides experimental evidence that an evolutionary algorithm is able to find update functions that generate simple patterns, with some form of self-healing. Miller et al. represent their update function as a boolean circuit, and optimise it with a variant of Genetic Programming, namely Cartesian Genetic Programming, also introduced by Miller [87]. Further works demonstrated that different kinds of function representations are also suitable to match the same simple patterns: Multi-Layer Perceptrons in Federici et al. [106], binary rules systems in Gordon et al. [38], Regulatory Gene Networks in Chavoya et al. [14, 13].

4.3.2 Cell Communications

In all the works presented here, the state variables of a cell are separated in two groups, the *internal chemicals* and the *external chemicals*[3]. The *external chemicals* are the state variables of a cell that are readable by its neighbours, whereas the *internal chemicals* are visible only by the cell itself. Miller [85, 86], Gordon [38], and Chavoya in his early work [12] don't use any *internal chemicals*, and all state variables are visible outside the cell, whereas Federici [106] introduces the idea of internal chemicals, followed by Chavoya in his most recent works [14, 13]. *External chemicals* have at least one clear role: communication. The internal chemicals then play the role of the memory of a cell. It should be noticed that such an internal memory is not mandatory, as external chemicals can also play this role implicitly. On the other hand, a low number of external chemicals reduces the number of inputs and outputs of the update function, which might reduce its complexity.

In Miller's work, the communication process includes a *diffusion* of the external chemicals. Miller implemented the diffusion of a given chemical as a low-pass filter (a mean filter with radius 1) applied over the map of this chemical. This can be seen as a very rough approximation of the continuous diffusion on a discrete space (though a Gaussian filter would be a more accurate approximation), and the remaining part of the update function as the chemicals reactions function. But similar approaches without diffusion (as in Federici's

[3] In some related works, the word *proteins* is also used, in lieu of *chemicals*

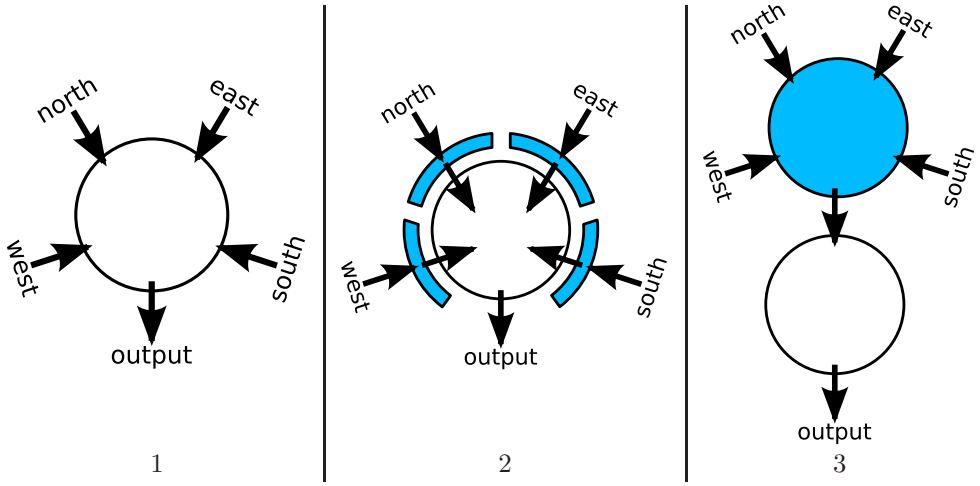


Figure 4.4: Symbolic representations of different proposals for the update function, that produce a new cell state from 4 neighbour states. The white circles represent the parts that are optimized by evolution, while the colored circles represent the ad-hoc user-defined parts. Existing works range from the more general one on figure 1, where the complete update function is evolved, to that of figure 2, where the neighbor states are pre-processed before being handled to the evolved part, and to the most ad-hoc one on figure 3, where pre-processing of all neighbor inputs results in a single input for the evolved function (as e.g. in totalistic cellular automata, where the ad-hoc part is a simple sum of its inputs).

work), and/or without internal states (as in Gordon's and Chavoya's works) are also capable of pattern regression. It should be also underlined that Gordon's and Chavoya's works do not address the issue of the self-healing. Studying the effect of a diffusion mechanism on an embryogenic flag setting might be fruitful and remains to be done, to the best of our knowledge.

4.3.3 Neighbourhoods and Inputs

The most widely used neighborhoods are the Moore and the Von Neumann neighbourhoods (see figure 4.2), with the exception of initial work by Chavoya [12], that uses the less common Margolus neighbourhood [115]. However, no clear advantage of one neighborhood over the other has ever been clearly demonstrated in any of the works reviewed here. The Moore neighbourhood is more demanding in terms of the number of inputs of the update function, but it might help reducing its internal complexity. Furthermore, all those works consider *non-totalistic* update functions: the states of all neighbour cells are accessible separately – with the exception of Gordon's work that uses a *totalistic* update function (only the sum of the neighbor inputs is given to the update function).

The totalistic approach greatly reduces the number of inputs of the evolvable part of the update function, but at the cost of great loss of information: the impact on the inner complexity of the update function is yet unknown.

4.3.4 Synchronous and Asynchronous Updates

Both synchronous and asynchronous updates can be found in the literature. To the best of our knowledge, only Chavoya uses a random asynchronous updates [12]. Adding randomness during the development process might enforce the robustness of the pattern formation mechanism with regards to random perturbation. On the other hand, it might also increase the difficulty of the problem: this issue remains to be studied in detail. All other works use either fully synchronous updates, or a mixture of synchronous updates for the external chemicals and deterministic asynchronous updates for the internal chemicals (in Miller's work). Here again, though asynchronicity can make a difference in the properties of a cellular automaton [31], a precise study of its influence in the context of the embryogenic flag problem remains to be done.

4.3.5 Initial Setup

Miller initializes cells in the *dead state*, and all the chemicals concentrations to the same value. The only exception is a single alive cell, in the middle of the embryogenic flag. Federici, and Chavoya in his early work, both use the same initial setup. With such an initial setup, the embryogenic flag is almost in an homogeneous state.

In contrast, other works by Chavoya set up the chemicals such that they form two orthogonal gradients, while also starting the development with a single alive cell. One might argue that such an initialisation simplifies a lot the problem, since, as discussed in Section 4.2, it turns the embryogenic flag problem into a simple regression problem.

Gordon also uses non-homogeneous initial conditions. Because he uses totalistic update functions, all cells having symmetrical inputs will have exactly the same behavior. Hence, starting from homogeneous starting condition would bias the search to find only symmetrical patterns. Hence using non-symmetric starting condition is the only way to break this intrinsic symmetry. However, the non-symmetric starting conditions used by Gordon are much less informative than the gradients for the chemicals.

4.3.6 Birth, Growth and Death

In all the works that are surveyed here, the authors introduce a cell life cycle mechanism. In Miller's work, a cell can be either dead or alive, and this is stated by a single bit of the cell state. Both dead and alive cells update their external chemicals, while only alive cells update their internal chemicals. A living cell can turn alive its dead neighbours, and can also kill itself. A very similar setting

is employed by Federici. On the other hand, in Chavoya's work, no diffusion of the external chemicals is done, so dead cells are completely inactive.

On the opposite, there is no idea of dead or alive cells in Gordon's work, and all the cells are updated at each time steps. In that sense, this approach is close to the reaction-diffusion systems (see Section 4.2.4). Without birth and death mechanism, the optimizer is nevertheless able to find update function that fits target patterns well. Moreover, as will be demonstrated in Sections 4.5 4.6 and 4.5, in the work presented there, previously unseen levels of self-healing are achieved, which might be partly explained by the absence of birth/death cycles. It turns out that cells birth and death might not be a mandatory mechanism for pattern formation (and biological relevance is not an issue as far as Evolutionary Design is concerned). To the best of our knowledge, no work has been done to check the positive or negative effect of introducing a cell life cycle. Of course, one argument in favor of introducing such a cycle is the parsimony with respect to computing resources: dead cells consume much less computing resources, decreasing the cost of the update of the embryogenic flag. But as far as accuracy of pattern identification is concerned, the comparison remains to be done.

4.3.7 Stopping Criterion

One motivation behind cellular approaches is their promise of scalable solutions to a problem. In the framework of the embryogenic flags, it could be seen as the power to regress the same pattern at several scales. An essential step toward scalability is to use the same controller and the same initial state for all the cells: the genotype size is theoretically independent from the pattern size. But large patterns also mean longer development times. This is why, in our opinion, a meaningful answer to the embryogenic flag problem should provide a stopping criterion suitable for any pattern size.

One of the most simple stopping criterion for the development phase is to simply fix a number of development steps. This is what is done by Miller, Federici, and Chavoya in some of his work [13]. There is then little chance indeed that the same number of development steps can be used for different scales of the same pattern (say 16×16 and 32×32), thus hindering the possible scalability of these approaches.

In [12], the number of development steps is adjusted by the optimization itself. However, here again, a drawback of this approach is that if an update function has been optimized to match a 16×16 pattern, it can not be reused as such to match the same pattern with a higher resolution, e.g. 32×32. A new optimization must be performed to find the suitable number of development steps (though the previous solution for the 16×16 pattern might be a good initial guess).

Gordon et al. [38] check if a fixed point or a limit cycle was reached, and stop the development anyway if none of those two conditions are met after a fixed number of steps. In such case, a direct reuse is possible without a new optimization, at least in principle, since no number of steps is enforced.

Although Federici uses a fixed number of steps for the development, his

work features an original development control mechanism. A candidate embryogenic flag is stopped after 12 update steps from it initial state. Each of the 12 steps is performed by a different update function, but those 12 update functions are optimized separately in an incremental way. At the beginning of the optimization, a single update function is used for each step. After several optimization epochs, a second update function is introduced, as a clone of the first one. The first, original update function is no longer modified by the optimizer, it only evolves the second one. The first update function is in charge of the first development steps, whereas the second update function is in charge of the remaining development steps. This process is repeated, and several update functions, termed *stages* by Federici, are introduced during the optimization. A comparative study demonstrates an increasing target pattern fitting score when increasing the number of stages.

A striking trait of the state of the art work on the embryogenic flag problem is the number of development steps used. In Miller's work, the dimension of the target patterns is 16×16. The initial state of all the cells is the same, except for the central cell. The development is stopped after 9 steps: this means that a signal starting at the center can traverse pattern square at most once. This has a second effect: the development dynamic is kept simple, which might reduce the complexity of the reachable target patterns. Hence it reduces the expressivity of the system. Similarly Federici used 12 development steps for 32×32 targets. Note that Chavoya [13] uses 100 development steps for 33×33 targets, leaving room for somewhat longer and more complex dynamics to emerge.

4.3.8 Self-Repairing Properties

Miller performed an a posteriori analysis of the developmental processes carried out by his approach. During the controller optimization, the pattern generated by a developmental process was evaluated after a fixed number of steps. Nevertheless, some of his solutions reached a steady-state in the developmental process, sometimes acting as loose attractors: after random perturbations, the development would produce some patterns very similar to the initial target pattern (the 3-bands French flag).

4.3.9 Relevance to Evolutionary Design

At first sight, 2D pattern regression might seem a bit far from Evolutionary Design. However, whereas the cell colours are interpreted as an actual colour, for convenience, they could be considered from other points of view, such as as logic gates in the case of boolean circuit design for instance. Indeed, in [67], Kowaliw et al. use an embryogenic flag approach, where the cell colors are associated to structural truss patches (more details in chapter 5). This is still an embryogenic flag problem, but of course the fitness of an expressed pattern is no longer the similarity with a target pattern, but an adequacy measure from Solid Mechanics. Kowaliw et al. show the inherent scalability of the approach, as well as its ability to express a pattern within different contexts. A square lattice could

	Julian Miller et al.	Timothy Gordon et al.
states	binary string	binary string
neighbourhood	8	4
update function representation	Boolean logic circuit and diffusion	implicit boolean transition table
cell states update order	asynchronous, deterministic order	
ontogeny control	fixed number of development steps given by user	stop when steady-state reached and fixed number of development steps given by user

Table 4.1: Major traits of some works around the embryogenic flags reviewed in this section

	Diego Federici et al.	Chavoya et al. 2006	Chavoya et al. 2007
states	real valued vector	1 bit	1 bit and 2 reals
neighbourhood	8	8, Margolus	8
update function representation	Multi Layer Perceptron and diffusion	explicit boolean transition table	Gene Regulatory Network
cell states update order	asynchronous, cell age order	asynchronous, random order	asynchronous, random order
ontogeny control	development carried through stages, fixed number of development steps given by user	fixed number of development steps taken from genotype	fixed number of development steps given by user

Table 4.2: Major traits of some works around the embryogenic flags reviewed in this section

be seen as a constraint, and indeed, many of the principles developed within the embryogenic framework, as diffusion, external and internal chemicals, seem applicable on less regular structures such as graphs. Regression of a pattern by cells on less regular structures than a square lattice would be a step forward in sense.

A crucial issue that we discussed in chapter 3 is the coupling of an Evolutionary Design representation with the physical phenomenon underlying the design problem. In the works introduced here, no such closed loop is present. Nevertheless, some reaction-diffusion systems have been coupled with physical phenomena for biological phenomena modeling, as introduced in chapter 6 of [89]. Such closed loop systems are able to express stable and self-repairing patterns, so at least the idea seems reasonable. A benchmark problem would allow constructive trials and comparisons. Truss structures could provide such a testbed (and indeed we do experiment with cellular representations for truss structures in chapter 5), but it might be a bit too technical. Heat diffusion and mean temperature minimisation problem, as introduced by Estevez et al. [29] fits perfectly with the embryogenic flag problem and is pretty much straightforward to implement.

4.4 Methodology

The experiments described in the remaining of this chapter focus on representations of the update and expression functions, and on the influence of the stopping criterion of the development. This section introduces the methodology used for these experiments.

4.4.1 The Target Patterns

Four different target patterns, shown on the figure 4.5, have been used. They are grayscale 32×32 patterns, where the pixel values are coded in $[0, 1]$ (0 for black, 1 for white).

The *two bands* target is even simpler than the traditional *French Flag* pattern used in many of the previous works: only two bands of same size, with maximum contrast. This pattern perfectly fits the square lattice of the cells, and thus should be really easy to match. It is used here as a baseline experiment: if a given approach has difficulties matching this pattern, one might suspect a deep flaw. The *three bands* target is very similar to the traditional *French flag*, but the bands have here different widths. Again, this pattern perfectly fits the square lattice of the cells, so it should not be very difficult either. The *disc* target is a discretized disc, that will certainly prove more difficult to regress than the bands patterns, with cells that are embedded on a square lattice. Finally, the *half-discs* target combines two half-bands with a discretized disc, and we except this pattern to be rather hard to match.

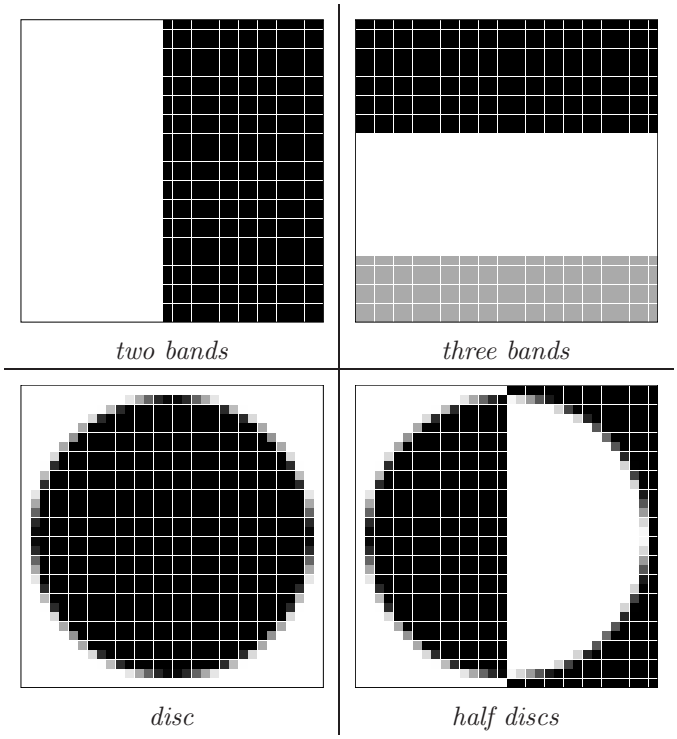


Figure 4.5: The four 32×32 targets used for the experiments

4.4.2 Stopping criterion

When working with any developmental approach, a development stopping criterion has to be chosen. In most previous approaches (see Section 4.3.7), the number of iterations is fixed once and for all by the programmer. This might seem crude. Indeed, it is clear that the number of iterations that are necessary to reach a given state depends on the state itself, but also on the organism. Moreover, it should most probably also depend on the experimental conditions: the dimension of the grid, the number of chemicals, ... Because there seems to be no general way to a priori determine the number of iterations that should be allocated to the organisms to reach a target phenotype, a good solution is probably to leave this parameter free, and to let it change along evolution.

Furthermore, we are mostly interested in stable patterns, the ones that are expressed by a steady state of the embryogenic flag, with the largest possible basin of attraction to ensure some self-healing properties. In order to detect such a steady state, rather than adding it as yet another parameter to tune for the evolutionary algorithm, as in [12], we propose the following heuristic. At each development step, an *energy measure* of the embryogenic flag is computed. If the energy of an embryogenic flag remains the same over some time window, we are likely to have reached a steady state. A both simple and general energy is the sum of the squared state values of all the cells. However, due to the numeric inaccuracies of floating point calculus, perfect equilibrium will never be reached. A more robust characterization of a steady-state is the standard deviation of the energy over a time window. If the standard deviation is lower than some small enough threshold ε, we will consider that the embryogenic flag has reached a steady state and stop the development. In practice, a time window of size at least 16, with a threshold ε of 10^{-15} were found to be satisfactory: none of the cell controllers that matched this criterion were unstable while satisfying this criterion.

More formally, the proposed stopping criterion can formulated as follows: let $state(c,t)$ be the state of cell c (that can be a scalar or a vector), and $||.||_2$ the Euclidean norm on the state space. The energy of the organism at a given time t can then be defined by

$$e(t) = \sqrt{\sum_{\text{all cells } c} ||state(c,t)||^2}$$

The average energy over a time window of size k ending at time t is then defined as

$$\overline{e}(t,k) = \frac{1}{k} \sum_{i=t-k}^{t} e(i)$$

and the variance of the energy over the same time window is then

$$\mathcal{V}\text{ar}(t) = \sqrt{\frac{1}{k} \sum_{i=t-k}^{t} \left(e(i) - \overline{e}(t,k)\right)^2}$$

and the development process stops if $\mathcal{V}\text{ar}(t) < \varepsilon$.

However, there is absolutely no guarantee that a given controller will trigger this stopping criterion in a finite time. Indeed, some candidate controllers might lead to periodic or chaotic dynamics, and no steady state will ever be reached in such a case. Because there is no way to apriori predict the type of behavior of a given controller, an upper bound on the total number of development step has to be set anyway. But because our goal is to find stable pattern, i.e. a controller that reaches a steady-state, controllers that hit this upper bound will be considered uninteresting, and they will receive the worst possible fitness. For instance, for the 32×32 patterns considered here, we fixed this upper bound to 1024: this is long enough to allow a signal to traverse 32 times the embryogenic flag from one side to the other.

4.4.3 Pattern distance

Another important ingredient of an embryogenic flag setting is the pattern distance to be minimized so as to ensure that the expressed pattern of the embryogenic flag matches the target image. Designing a pattern distance that satisfies human subjectivity is not an easy task. For instance, we might want to consider that two patterns are very similar if they are different only up to a rotation or a translation. Detecting such cases requires sophisticated image analysis algorithms, and is definitely out of scope here – not to mention the computational cost. Because we want to express the difference between two sets of potentially high number of pixels with just a single value, some information loss is unavoidable anyway. We hence decided to use the raw Euclidean norm as a distance measure: let A and B be two $N \times M$ patterns whose pixel values belong to $[0, 1]$. Then the distance between A and B is

$$d(A, B) = \sum_{i=1}^{N} \sum_{j=1}^{M} (A_{i,j} - B_{i,j})^2$$

In order to normalize the fitness values, let us define, for any target pattern T:

$$d_{\max}(T) = \max_{X}(d(T, X)) = \sum_{i=1}^{N} \sum_{j=1}^{M} (\max(1 - T_{i,j}, T_{i,j}))^2$$

Then the following quantity takes its values in $[0, 1]$:

$$\mathcal{D}(T, X) = \frac{d(T, X)}{d_{\max}(T)}$$

and 0 means a perfect match between T and X while 1 is the worst possible mis-match.

This normalized distance has been used on all experiments detailed in the remaining of this chapter. It is not adequante to match our subjective view of pattern distance, but it is at least very cheap to compute. More important, albeit being not a very informative measure, it seemed to be enough to guide the evolution toward controller that matched the patterns presented used here.

4.4.4 Fitness function

Now that the target patterns, the stopping criterion for the development, and the pattern distance have been defined, we can summarize the complete evaluation process for a candidate cell controller:

1. An embryogenic flag, with the size of the target pattern, is setup. All the initial states are set to the same value, 0.
2. One cell state update is performed.
3. the energy $e(t)$ of the organism is computed, as well as its variance over the last 16 steps $\mathcal{V}\text{ar}(t)$
4. If $\mathcal{V}\text{ar} < 10^{-15}$, then we consider that a steady state has been reached and we proceed to step 7.
5. If more than 1024 development steps have occurred, we return the worst possible fitness value of 1.
6. Else the development process continues: go to step 2.
7. The distance $\mathcal{D}$ between the expressed pattern of the cells and the target pattern is returned as fitness value.

4.4.5 Steady state stability

Once a cell controller has triggered the stopping criterion based on the variance of its energy over a time window, because our goal is also to reach self-healing behaviors (see Section 4.3.8), we are interested in checking the stability of the steady state that has been reached. We will use a straightforward raw characterization of stability: If a steady state A is stable, a state B close to A should lead to state A after some development steps. On the other hand, a steady state A might be a unstable, meaning that any state B close to A will probably move away from A when further development steps are performed (see figure 4.6).

Practically, we will be adding to the state of each cell some Gaussian noise, centered on 0 and with standard deviation σ. A very stable steady state A will always return to A after some development steps despite high values of σ. Moderately stable steady states might be resilient to small σ values, but higher values of σ will lead the embryogenic flag to another, different steady state, or even to a completely different regime: oscillations or even chaotic behavior. Such verification will only be done a posteriori, not impacting the evolution.

4.4.6 Parametric optimization

In two of the three series of experiments that will be presented next, the genotype is a vector of real values. In such case, optimizing the controller amounts to parametric optimization, and we have used *CMA-ES*, the state-of-the-art algorithm for continuous evolutionary optimisation, introduced by N. Hansen et

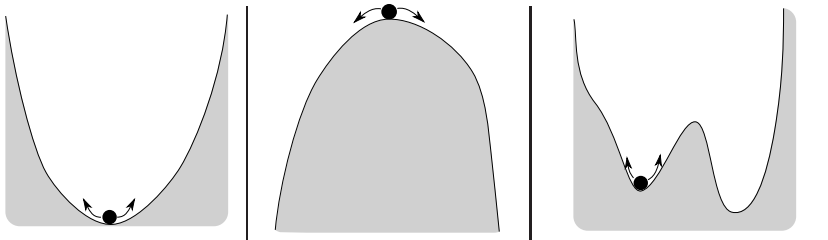


Figure 4.6: Example of a simplistic dynamical system: a ball is rolling on a 1D landscape, subject only to vertical gravity. When there is a single valley in the landscape (left), the ball will always return to the equilibrium point (the bottom of the valley), no matter how much you move it before releasing it. On the opposite, on a peaked landscape (center), any small perturbation of the ball position will result in a very large movement of the ball, away from the equilibrium point. On more complex landscapes like the one on the right, the steady state might be stable for small perturbations, but larger perturbations might lead to other steady states (or put the system into another regime).

al. [3]. *CMA-ES* main feature is its ability to adapt to the fitness landscape by learning online linear dependencies between the variables. Most of it parameters are adaptive, and in particular no parameter needs to be tuned, apart from the population initialization parameters and the initial mutation step size. For all the experiments presented in this chapter, the initial population was drawn from a Gaussian distribution centered on 0 and with standard deviation 10^{-1}, and the initial mutation step size was set to 10^{-1}.

4.5 Neural networks as update functions: NEAT

In this section, we show the first experiments that validated the approach to embryogenic flags proposed in the preceding Sections using universal approximators as update and expression functions. In order to do so, we have chosen Discrete Time, continuous state, Recurrent Neural Network with sigmoidal transfer functions as a cell controllers. The choice of Neural Networks was inspired by the long-known property that Neural Networks are Universal Approximators [52] ... and the reputation and availability of the NEAT algorithm.

4.5.1 The model

The cell states are real vectors with values are in $[0, 1]$. A cell state is defined by its N neural network neurons activations and M *chemicals* values. Initially, all the activations and chemicals are set to 0. The update function, applied in a synchronous fashion, takes as inputs the neurons activations of a cell, and the chemicals levels of its four neighbours (Von Neumann neighbourhood). It

returns the new activation levels and the new chemicals concentration values. If there are N neurons and M chemicals, the more general form of update rule at time step t for neuron i of a Discrete Time, continuous state, Recurrent Neural Network is

$$a_i(t+1) = \sigma(\sum_{j=1}^{N} w_{i,j} a_j(t) + \sum_{j=1}^{M} z_{i,j} I_j(t))$$

where $a_i(t)$ is the activation of neuron i at time t, $I_j(t)$ is the concentration of the j^{th} chemical at time t, $w_{i,j}$ is the weight of the connection from neuron j to neuron i (0 if no connection exists), $z_{i,j}$ is the weight of the connection from input j to neuron i, and $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic sigmoid function.

The fitness is computed according to the procedure described in Section 4.4.4: development stops if the variance of the energy over the last 16 steps is below 10^{-15}.

4.5.2 Controller Optimization

Even though the smaller class of simple sigmoidal 3-Layer Perceptron has the Universal Approximator property, determining the number of hidden units for a MLP remains an open issue, and practical studies [19, 39, 113] have demonstrated that exploring the space of more complex topologies (including recurrent topologies) could be more efficient than just experiencing with a one hidden layer perceptron. Moreover, many algorithms have been proposed for the evolution of Neural Networks. However, the *NEAT* algorithm [113] has recently emerged as performing very well on a wide range of tasks. It makes it possible to explore both feed-forward and recurrent topologies. This algorithm relies on a direct encoding of neural network topologies that are evolved using a non-standard evolutionary optimization scheme. The main feature of *NEAT* is that it explores the topologies from the bottom-up: starting from the simplest possible topology for the problem at hand, it performs variations by adding neurons and connections to the existing network in such a way that the behavior of the network is preserved as much as possible – this makes it possible to explore topology in a non destructive manner.

We re-implemented the *NEAT* algorithm, and validated our implementation from published results. For all the experiments in this paper, *NEAT* parameters have been set to match the results[4] given in [113] for solving the sample *XOR regression* and *double-pole balancing* tasks. Those values seemed robust for the problem at hand, according to a limited parametric study. Those parameters are summarized in table 4.3.

As already noted, an interesting feature of *NEAT* algorithm is that it can handle the evolution of both feed-forward and recurrent neural networks – hence enabling an easy comparison of both models. Another interesting feature of

[4]Finding those parameters required a rather long time for such an auxiliary task. Those parameters are not orthogonals, they have combined influences in the overall algorithm behaviour. Moreover, some parameters, like the fitness sharing parameters are not intuitive at all to setup.

Population size	500
Max. number of evaluations	250000
Reproduction ratio per species	0.2
Elite size per species	1
Crossover prob.	0.15
Add-node mutation prob.	0.01
Add-link mutation prob.	0.01
Enable-link mutation prob.	0.045
Disable-link mutation prob.	0.045
Gaussian weights mutation prob.	0.8
Std. dev. for Gaussian weight mutation	0.1
Uniform weights mutation prob.	0.01
Distance parameters for fitness sharing	1.0 – 1.0 – 0.2

Table 4.3: NEAT parameters

NEAT is that it allows the user to declare some constraints on the topology – in this case, we enforced that all input and output neurons are connected to at least one neuron in the controller.

4.5.3 Experimental Setup

In order to explore different models within the general context described in this section, four instances of the proposed embryogenic model are experimented with:

- a feedforward neural network, using 1 chemical, termed *1-ffwd*
- a recurrent neural network, using 1 chemical, termed *1-recurr*
- a feedforward neural network, using 2 chemical, termed *2-ffwd*
- a recurrent neural network, using 2 chemical, termed *2-recurr*

However, there are (at least) two possible causes of error in the proposed approach: on the one hand, there might not exist any fixed point of the multicellular developmental systems under study that can approximate the target image; but on the other hand, even if a good solution does exist, the chosen computation method (evolutionary optimization of a neural network using *NEAT*) might not be able to approximate it. Note that this situation is common to all computational approaches of complex systems: the former error is termed 'modeling error', and the latter 'method error'. A third type of error is also reported in numerical experiments, the 'numerical error', due to propagating round-offs, and will be neglected here.

In order to try to discriminate between the modeling error and the method error, a fifth model is also run, on the same test cases and with similar experimental conditions than the four developmental approaches described above: the

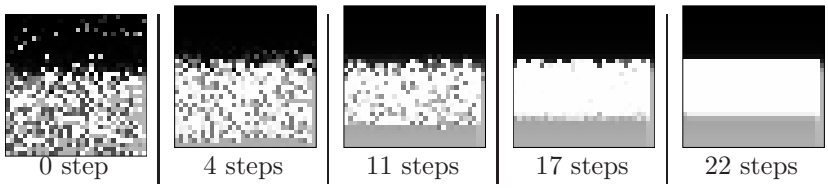


Figure 4.7: Self-healing on the three-bands problem for the recurrent NN and 2 chemicals

layout is exactly the same (a 2D grid of cells), the same *NEAT* parameters are used (to evolve a feed-forward neural network), and selection proceeds using the same fitness. However, there is no chemical nor any exchange of information between neighboring cells, and on the other hand, all cells receive as inputs their (x,y) coordinates on the grid. This a strict equivalent to the non-developmental solution of the embryogenic flag proposed in Section 4.2. In the following, the results of this model will be considered as reference results, as it is not expected that any developmental approach can ever beat a totally informed model using the same *NEAT* optimization tool. This experiment is termed *non-dev* from now on.

All five models described above have been run on the four target patterns showed on figure 4.5. All results presented in the following are statistics gathered over 16 independent runs.

As already said, the evolutionary neural network optimizer is *NEAT*, with the settings that are described in Table 4.3. It is worth noticing that during all runs, no bloat (ie. uncontrollable structure growth with no functional advantages) was ever observed for the *NEAT* genotypes. The mean size of the networks (measured by the total number of edges between neurons) gently grew from its starting value (between 5 and 10 depending on the model) to some final value below 40 – the largest experiment reaching 45. This first result confirms the robustness of this optimization tool, but also, to some extent, demonstrates the well-posedness of the problems *NEAT* was solving (bloating for Neural Networks can be a sign of over-fitting ill-conditioned data).

4.5.4 Results and discussion

All the four settings of the developmental model were able to provide stable development processes for the four targets. The robustness of the fixed points was checked by applying a centered Gaussian perturbation with unit standard deviation to the states of all neurons. The good news is that for all perturbations, 100% of the feedforward controllers and 75% of the recurrent controllers return exactly to the same state they had before the perturbation. The other 25% recurrent controllers return to a steady state that is very close to the one they had before the perturbation. An example of perfect and fast self-healing for the *three-bands* problem is shown in figure 4.7.

Because all organisms are allowed 1024 iterations in their growth process, it could be feared that several hundreds iterations would be needed before stabilization even for the best solutions found by the algorithm. The total computational costs would henceforth have been tremendously higher that it already is. The good news is that in all cases, and for all embryogenic models, the whole population rapidly contains a large majority of organisms that do stabilize within a few dozen iterations. For the easy *three bands* problem, around 40 iterations are enough, whereas for the more difficult *half discs* problem, about 120 iterations are needed.

The *two bands* (figure 4.8) target seemed a really easy problem: the convergence is very fast for the four settings, quite close to the convergence featured by the non developmental approach. Three of the four models reached a pattern matching score of 10^{-3}, which is a visually convincing matching.

The story is less idyllic with the *three bands* target (figure 4.9): the median best solutions does not reach the 10^{-3} matching score, although some lucky solutions did. All the 4 settings tends to lead to optimization that stalls after 100000 evaluations, around a 2×10^{-2} matching score. In contrast, the non developmental approach can still reach a pattern matching score of 10^{-3} in less than 150000 evaluations. Thus, the *three band* target is a somewhat harder problem, whatever the approach.

Things even get harder with the *disc* target (figure 4.10). The non developmental is no longer able to reach the pattern matching score of 10^{-3}: the median score is above 10^{-2} after 250000 evaluations, though the runs did not stall yet. The developmental approaches reached a matching score between 10^{-1} and 8×10^{-2}, thus, pretty far from the non-developmental approach. Also, their optimizations runs progress much more slowly than for the *two bands* and the *three bands* targets.

The *half discs* is by far the most difficult target (figure 4.11). The non developmental approach reaches a matching score a little above 10^{-2}, whereas the developmental approaches do improve a lot the initial candidates solutions, however stalling at 2×10^{-1}

The coordinate-informed non-developmental approach always provides better matching scores with less evaluations. This is of course what was excepted, since the problem solved through development is by far harder, as exposed in Section 4.2. The non-developmental solutions is a yardstick of the difficulty to match a given pattern using a given optimizer. The four targets we choose provide problems with a gradual difficulty.

This developmental model based on *NEAT* is able to represent stable development processes, and is fairly well compatible with an optimization process. The approach used here is a rather simple and naive one. The cell controller is a general purpose neural network, and the communication between cells is achieved by simply reading parts of the neighbours states. Despite this rusticity, the evolutionary algorithm we used, *NEAT*, was able to find satisfying results on three of the four target patterns. Most of the controllers we found featured stable fixed point. On the other hand, we had real difficulties to correctly setup *NEAT*. Most of the *NEAT* parameters are dependent of each others, increasing

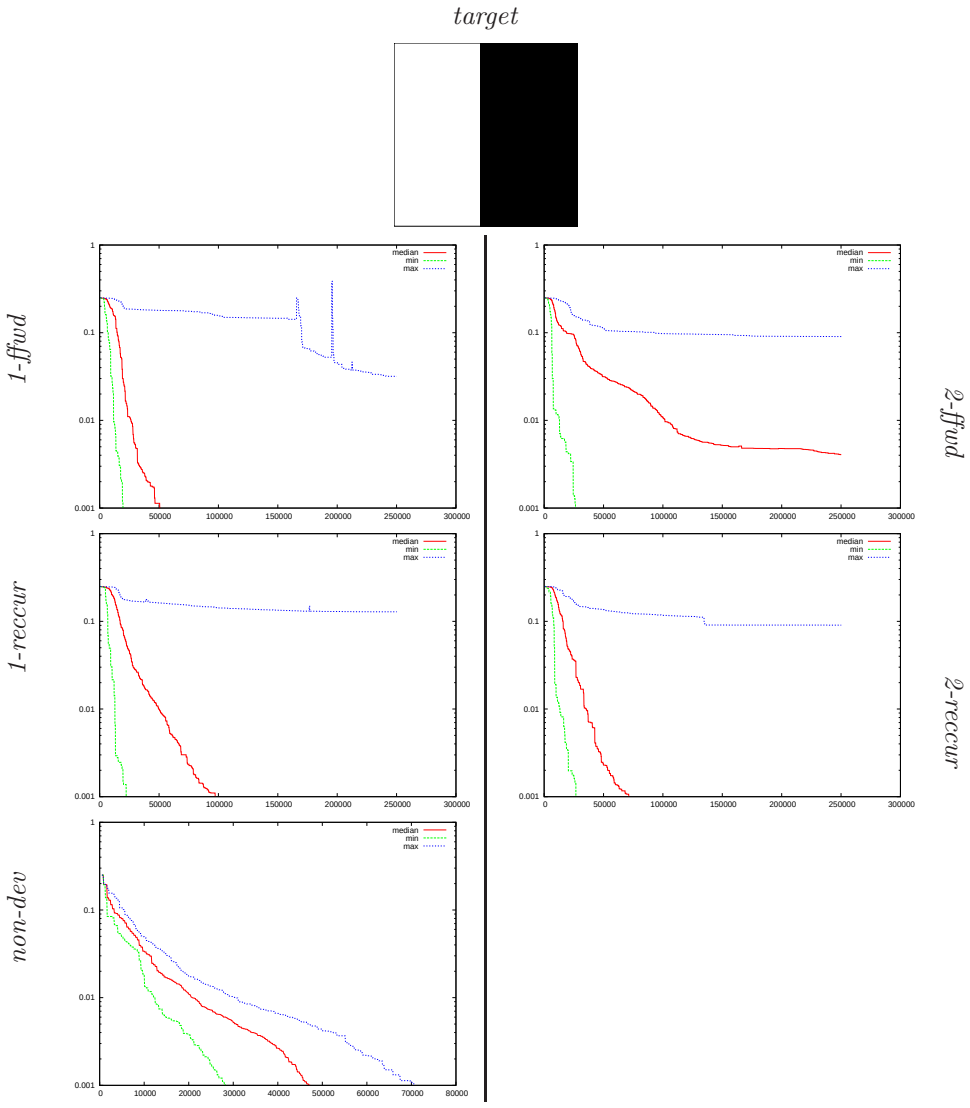


Figure 4.8: *NEAT* based approach: median, minimum and maximum fitnesses along evolution over 16 runs, for the *two-bands* pattern. Vertical axis is the fitness in log scale, horizontal axis is the number of evaluations.

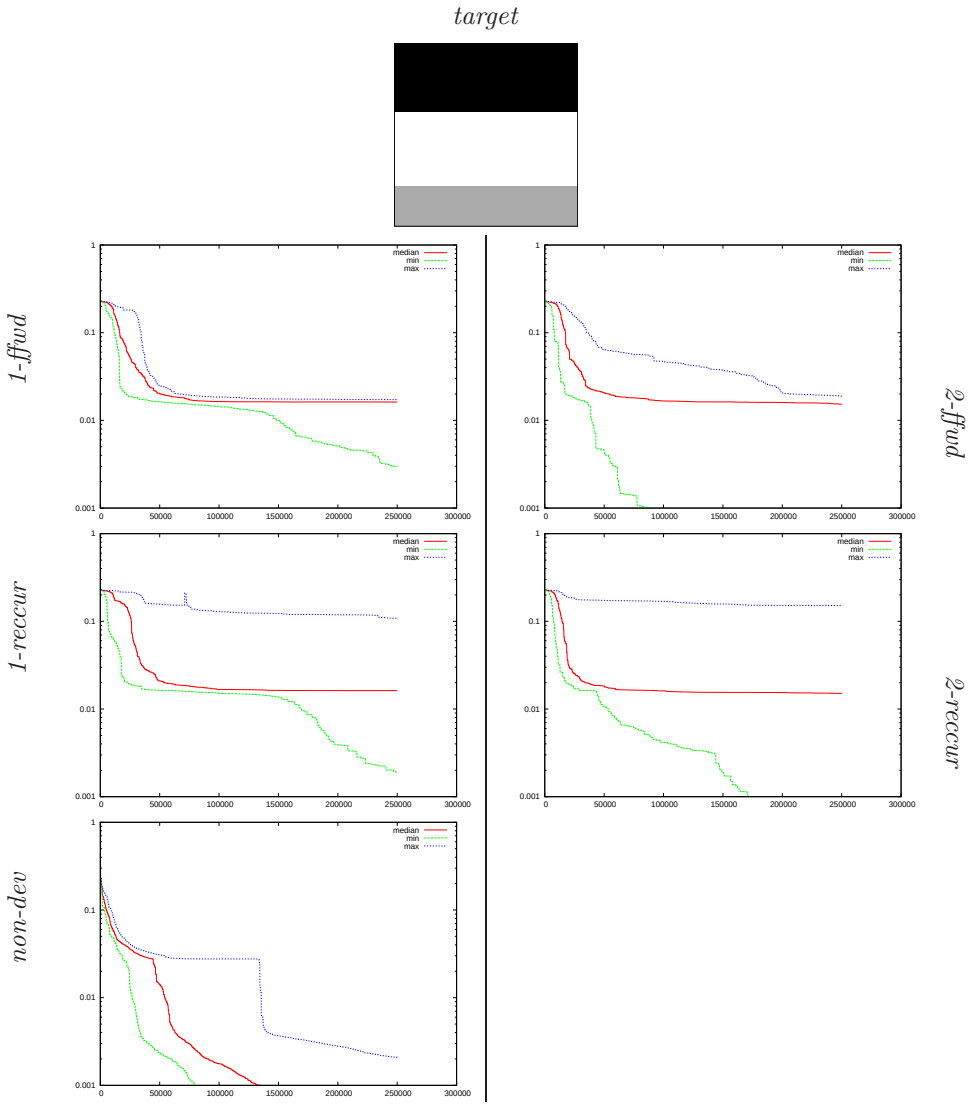


Figure 4.9: *NEAT* based approach: median, minimum and maximum fitnesses along evolution over 16 runs, for the *three-bands* pattern. Vertical axis is the fitness in log scale, horizontal axis is the number of evaluations.

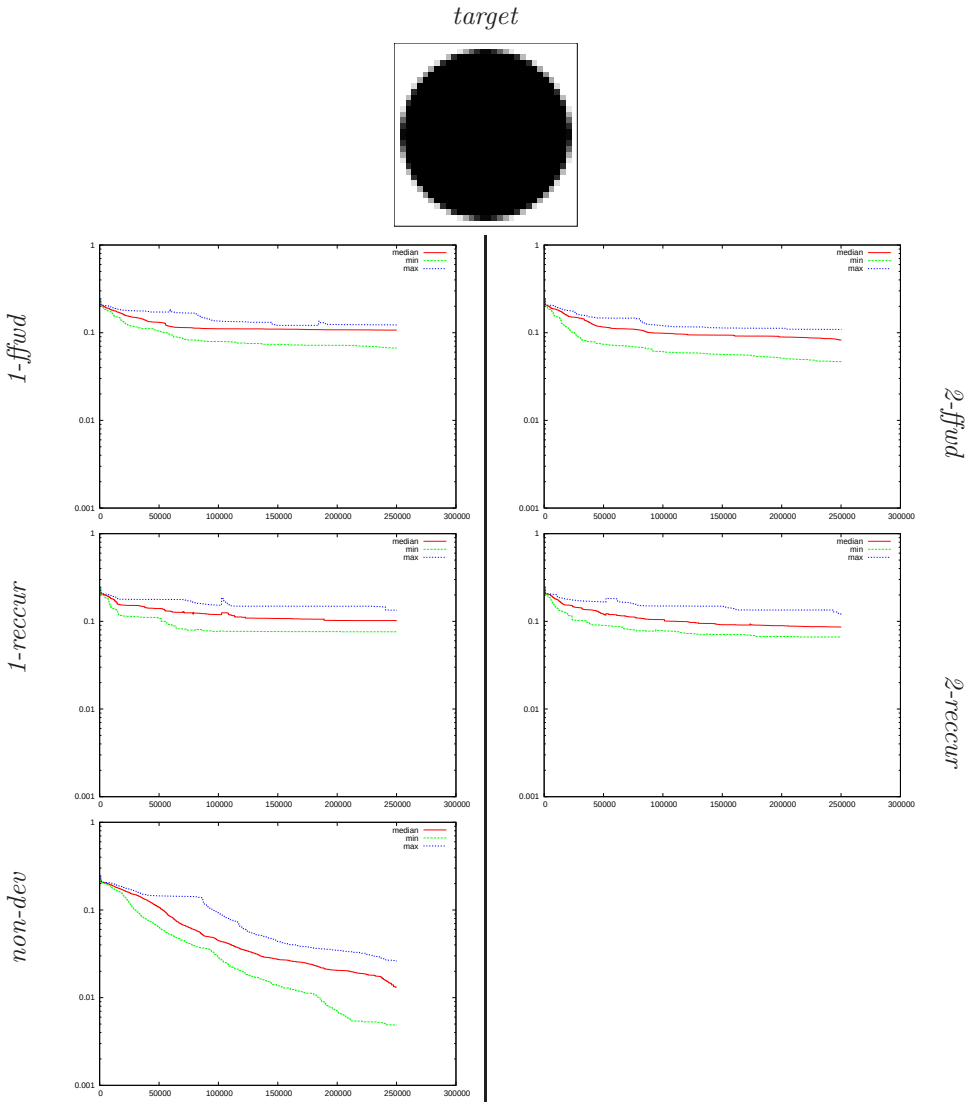


Figure 4.10: *NEAT* based approach: median, minimum and maximum fitnesses along evolution over 16 runs, for the *disc* pattern. Vertical axis is the fitness in log scale, horizontal axis is the number of evaluations.

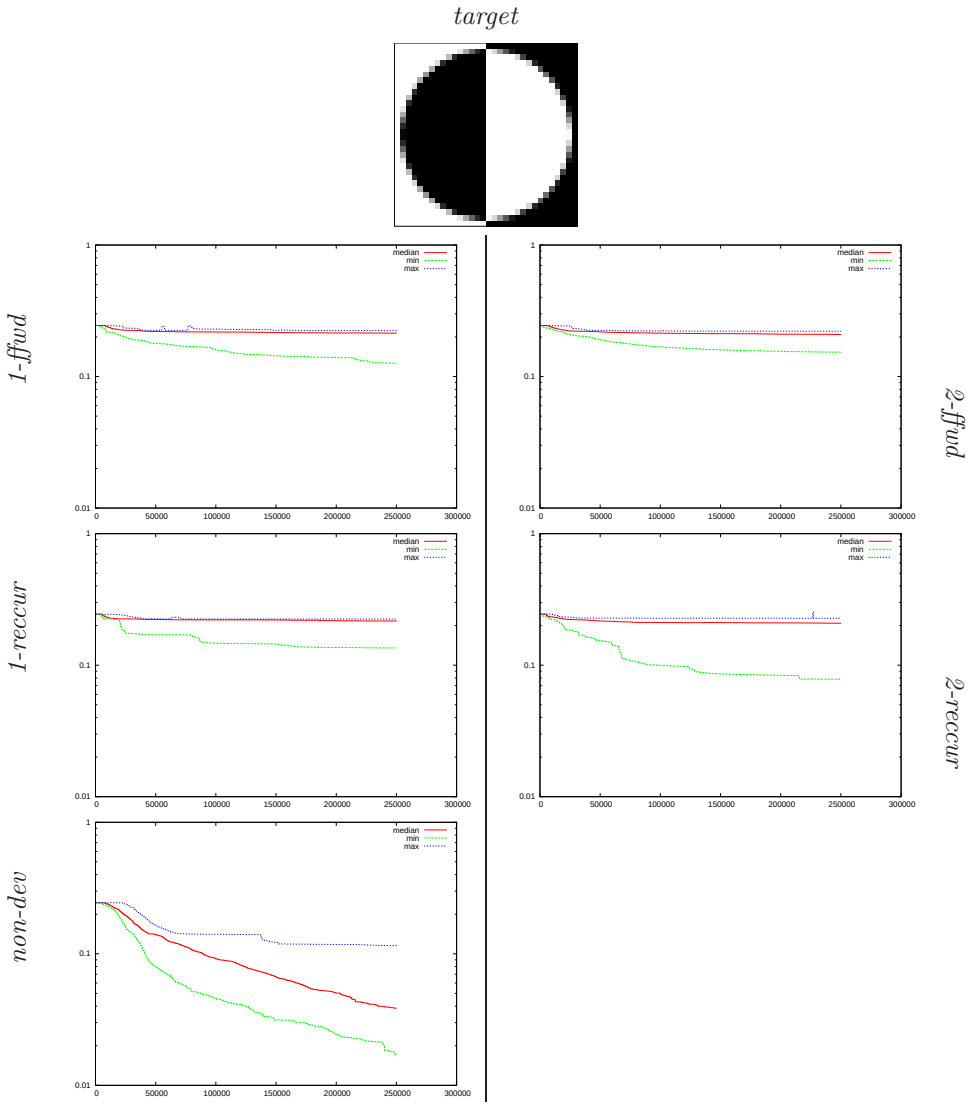


Figure 4.11: *NEAT* based approach: median, minimum and maximum fitnesses along evolution over 16 runs, for the *half-discs* pattern. Vertical axis is the fitness in log scale, horizontal axis is the number of evaluations.

the difficulty of the optimizer tuning, on top of the relatively high number of parameters. Moreover, the neural networks produced by *NEAT* seem much more complex than they should be, in regard to the simplicity of the target patterns. Maybe this unnecessary complexity is due to bad choices of *NEAT* parameters.

4.6 Reservoir computing for update functions

In order to try to circumvent the difficulties mentioned above, we propose here to replace the general recurrent neural network optimized with *NEAT* by an *Echo State Network* (ESNs) optimized by CMA-ES. *Echo State Networks* are a particular subset of general recurrent neural network, that retains the capacity of being Universal Approximators. Also, ESN come with some dynamical stability properties. Their topologies and a large part of their connections weights are drawn from a random distribution, entirely controlled by a few global numerical parameters. Thus, the problem of finding a controller for an embryogenic flag amounts to a fully parametric optimization problem. The parameters that remain to be decided by the user are the global parameters describing the ESN properties, and the specific parameters of the optimizer. As will be seen in the forthcoming short introduction to Echo State Networks, ESNs require fewer parameters to choose. Furthermore, by choosing an almost parameter free optimizer, we obtain a new approach to embryogenic flags that is easier to tune without introducing further complexity.

4.6.1 Echo State Networks

Echo State Networks have been proposed by Jaeger in 2001 [54, 55] with the objective of endowing a neural network with rich dynamic behavioural patterns while keeping learning complexity at a low level. An ESN is a discrete time, continuous state, recurrent neural network using a sigmoidal activation function for all neurons. A typical ESN is shown in figure 4.12, and will be used in this work: the input layer is totally connected to the hidden layer, both the hidden and input layers are totally connected to the output layer. Moreover, the output layer can be connected backward to the hidden layer. In this setup, the hidden layer, or *reservoir*, is randomly generated: N neurons are randomly connected up to a user-defined density of connections ρ. The weights of those connections are randomly set uniformly in $[-1, 1]$, and are then scaled so that the spectral radius of the connection matrix is less than a given value $\alpha < 1$, ensuring that the network exhibits the "echo state property", i.e. stays out of the chaotic behaviour zone whatever the input sequence (see e.g. [56]). The random construction of an ESN is thus determined by the 3 parameters N, ρ and α.

The main point in ESN is that only the weights going from the input and hidden nodes to the output nodes are to be learnt. If the problem has K inputs and L outputs and a reservoir of size N, this amounts to $(K + N) \times L$ free parameters. Moreover, in the case of supervised learning, where examples of

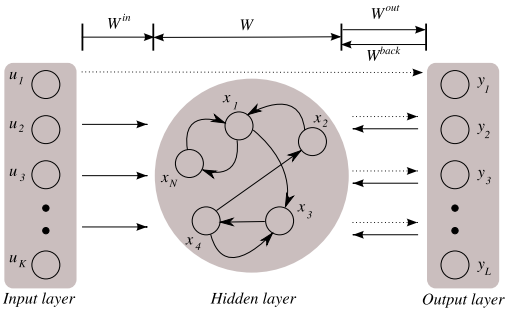


Figure 4.12: Schematic view of an Echo State Network. Plain arrows stand for weights that are randomly chosen and remain fixed, while dashed arrows represent the weights to be optimised, $(K + N) \times L$ where N is the number of neurons in the reservoir, K the number of inputs and L the number of outputs.

input-output values are given, the learning problem is reduced to a quadratic optimisation problem that can be efficiently and quickly solved by any deterministic optimisation procedure, even for very large values of N. In some sense, an ESN can be seen as a universal dynamical system approximator, which linearly combines the elementary dynamics contained in the reservoir [92]. ESN have been shown to perform surprisingly well in the context of supervised learning, in particular for problems of prediction of times series, though it has also been successfully used in the context of supervised robot control learning [57] and reinforcement learning of control tasks [59].

4.6.2 Controller optimization

The idea is to replace the general recurrent neural network in the setting introduced at Section 4.5 by an ESN. The developmental model remains the same, including the development stopping criterion (see Sections 4.4.4 and 4.3.7). An ESN is defined by the parameters of the reservoir generation, and its tunable weights. We will use here reservoirs of sizes 10 and 20, with connection densities of respectively 50% and 10%. In both cases, the damping factor (spectral radius) is set to 0.9. In the following , these settings are referred to as 10- and 20-ESN respectively.

When tackling a problem that does not pertain to supervised learning, as in the case of embryogenic flags, it is rather straightforward to replace the default learning algorithm of ESN with any derivative-free optimisers, such as Evolution Strategies or Simulated Annealing, to train an Echo State Network [4]. Yet, to date, and despite its intrinsic properties, ESN has not been applied to unsupervised problems with explicit criteria. In order to address this class of problems, this unsupervised learning task is turned into an optimisation task: optimising an ESN amounts to optimising a real-valued vector representing the plastic weights of the network (from inputs and reservoir to outputs (see figure

4.12). As stated in Section 4.4.6, we will use here *CMA-ES*. As in previous Section, 16 optimization runs will be run per setting and per target, with at most 250000 fitness evaluations.

4.6.3 Results and discussion

The results of the optimizations runs are shown on figure 4.13. The ESN based development processes features the same self-healing properties found with the *NEAT* based ones. Although the pattern matching scores are nearly 1 order of magnitude smaller for the *disc* pattern compared to those of the *NEAT* approach (as reported in Section 4.5.4), the scores are similar on the *half-discs* targets. Indeed, on this difficult target, the ESN based approach demonstrates a lower standard deviation, meaning that many optimizations of a *NEAT* based development processes are more likely to produce at least one well performing process.

Thus, ESN optimized by a Evolution Strategies can be seen as drop-in replacement of the *NEAT* optimized neural network, in the framework of the development model proposed in this work. On the one hand, no tedious parameter tuning seems to be necessary, as *CMA-ES* performs well with its default parameters, whereas *NEAT* does not features such an ease of use and is highly sensitive to its parameters settings. On the other hand, when using ESNs, most of the smart work is blindly delegated to the randomly initialized recurrent neural network, and as a consequence, to the optimizer. By breaking down the recurrent neural network into several parts, according to a fixed, hand-made architecture, we could probably reduce either the difficulty of the optimization task or the complexity of the controller. This is what we propose on the next section, while nevertheless keeping a fully parametric model (and hence being able to use the parameter-free CMA-ES).

4.7 A diffusion based approach

In the cell controller models introduced in previous Sections, the update and expression functions are entirely tuned by the optimization process. Those functions were represented by a single, opaque neural network. This approach has the advantage of simplicity. However, if any common underlying mechanism or principle exists for the embryogenic flag, it will have to be rebuilt at each optimization run. By breaking the single neural network into multiple functional units, we might relieve the optimizer of work.

One of those potentially general mechanisms is *diffusion*, as used by Miller et al. in [85]. Diffusion can be used to construct linear gradients, radial gradients and even more complex patterns, like it has been demonstrated with reaction-diffusion system. The diffusion process is chosen such that our model can mimic a discretized reaction-diffusion system. Experimental results will demonstrate the effectiveness of such a modification.

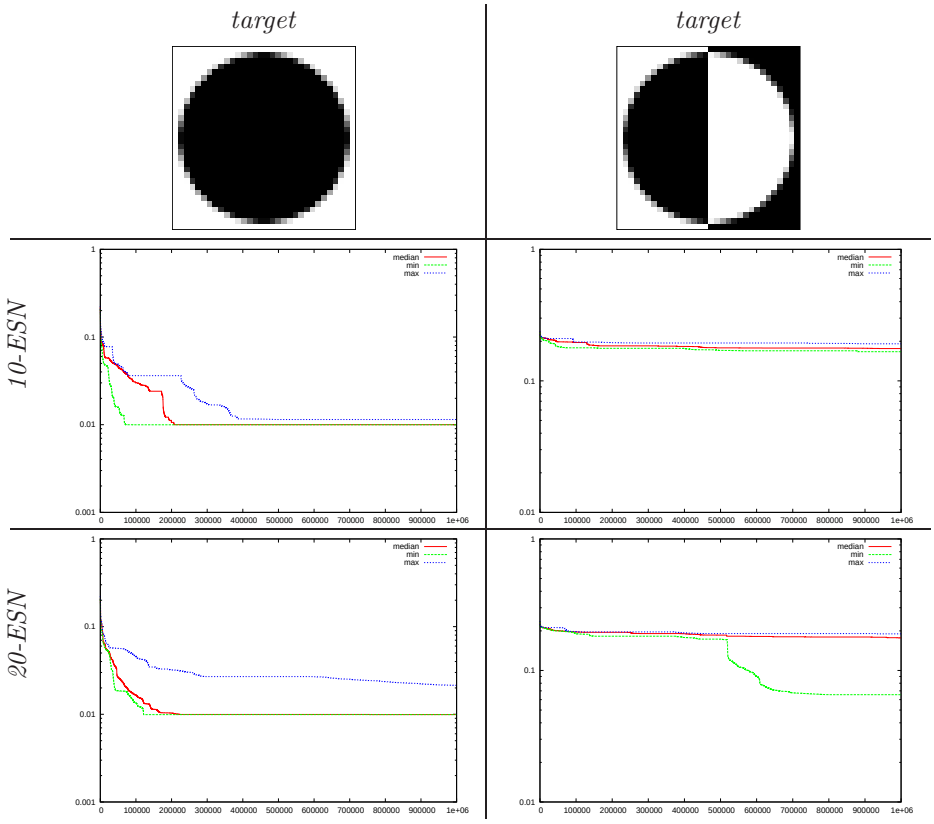


Figure 4.13: ESN based approach: median, minimum and maximum fitnesses along evolution over 16 runs, for the *disc* and the *half-disc* patterns. Vertical axis is the fitness in log scale, horizontal axis is the number of evaluations.

4.7.1 The model

A cell state is a pair of two vectors (u, v). u is the vector of external chemicals, and has M elements. v is the vector of internal chemicals, and has N elements. The values of all coordinates of u and v are in $[-1, 1]$, and are initially set to 0.

The update function is applied to the cell states synchronously. Its inputs are the internal chemicals of a cell, and the external chemicals of its 4 neighbours (Von Neumann neighbourhood). The update function outputs the new internal and external chemicals values, as follows:

$$\Delta_{i,j}(t) = f_u\left(u_{i+1,j}(t), u_{i-1,j}(t), u_{i,j}(t), v_{i,j}(t), u_{i,j-1}(t), u_{i,j+1}(t)\right)$$
$$u_{i,j}(t+1) = \tanh\left(u_{i,j}(t) + \Delta_{i,j}(t)\right)$$
$$v_{i,j}(t+1) = f_v\left(u_{i+1,j}(t), u_{i-1,j}(t), u_{i,j}(t), v_{i,j}(t), u_{i,j-1}(t), u_{i,j+1}(t)\right)$$

The functions f_u and f_v are both encoded as a Perceptron with one hidden layer, and with the hyperbolic tangent as transfer function. If the Perceptron has H hidden units and one bias input, the update function has $H(4M + N + 1) + (H+1)(M+N)$ parameters. In order to update the cell states, the external chemicals of each cells are subject to a Gaussian blur of a radius 1. Such a low-pass filter is a rather crude but is also an efficient implementation of a diffusion process. When a cell neighbour is missing, e.g. for the leftmost or the rightmost cells, the missing inputs are set to 0.

The expression function ϕ takes a cell state and returns a value in the $[0, 1]$ range, interpreted as gray level, according to the following equation:

$$\phi(u, v) = \frac{1}{2}\left(1 + \tanh\left(s_u \cdot u + s_v \cdot v + s_{bias}\right)\right)$$

This is indeed a single layer Perceptron with one bias input. s_u, s_v and s_{bias} are its synaptic weights. As a consequence, the expression function has $M + N + 1$ parameters.

Such a model can be viewed as a time discretized and space discretized reaction-diffusion system. Here, the diffusion is approximated by the Gaussian blur pass, and all the chemicals have the same diffusion constant. Applying a Gaussian filter to a chemical map is equivalent to perform a Weierstrass transform of this chemical map, which in turn is equivalent to a diffusion process with a diffusion coefficient of 1 [122]. The update function plays the role of the local chemical reaction function. The update function also plays the role of the Laplacian operator, thus it can tweak the diffusion process, to allow anisotropic diffusion.

The stopping criterion for the development phase, as defined in Section 4.4, is based on an energy measure of the whole embryogenic flag. For this diffusion model, the energy of a cell is defined as follows:

$$E(u, v) = u \cdot u + v \cdot v$$

The energy of a whole embryogenic flags is the sum of the energy of all its cells.

4.7.2 Controller Optimization

In the experiments presented here, the cell states are made of 1 internal and 2 external chemicals. This is much smaller than the 10 to 40 states per cell employed in our previous control models. The update function uses 8 hidden units. Thus, the genotypes are made of 111 real values (107 for the update function and 4 for the expression function). As in previous Sections, if a candidate solution fails to meet our stopping criterion after 1024 development steps, then it receives the worst possible fitness, 1. All the settings are tested with 32 optimization runs, which are stopped after 10^5 fitness evaluations. There are very few parameters to tune with this approach: M, N and H which controls the complexity (thus the compacity) of the model.

4.7.3 Results and discussion

The median, maximum and minimum of the fitness over the number of evaluations for the 4 target patterns are shown on the figure 4.14. The curves obtained by the diffusion-based approach are much more jaggy than those of the Neural Network based approaches, since the best current generation fitness is plotted, not the best ever fitness.

On the simplest problem, the *two bands* target, our new model does not have the strong bias of the *NEAT* or ENS based approaches. This is easily explained: the Perceptron used as an expression function in our model relies on a fixed transfer function, the hyperbolic tangent. This function is a rather gentle steep function, so obtaining sharp gray levels transitions requires a lot of fine tuning from the optimizer. On the opposite, *NEAT* can build very steep functions by linking neurons in long chains. Adding a steepness parameter to the expression function of our model might help to solve this issue. That bias does not appear on the *three bands* target runs, despite the sharp edges. The third gray band is difficult enough to build from the very poor information brought by the fitness function (a single number to compare 1024 pixels arrays), and all approaches became pretty much equal. On the *disc* target, the diffusion-based model obtains better results, which is explained by the fact that a circle is easy to build from an isotropic diffusion. Finally, better results are also obtained for the *half discs* target.

Overall, except for the *two bands* target, the diffusion-based model optimizations reaches better pattern matching scores. It also needs fewer evaluations than the optimizations carried with both previous approaches to reach a given pattern matching score. Where the previous approaches need 10 to 40 states per cell, the diffusion based approach reaches similar matchings with only 3 states per cells. Moreover, the diffusion based model relies on a purely parametric, almost parameter free, optimizer. The diffusion-based model seems to be clearly superior to the two previous approaches, as show on figure 4.15. An example of result patterns are shown in figure 4.17.

All the best controllers found with the diffusion-based model provide stable development. Low-level (standard deviation of 10^{-1}) of gaussian noise added to

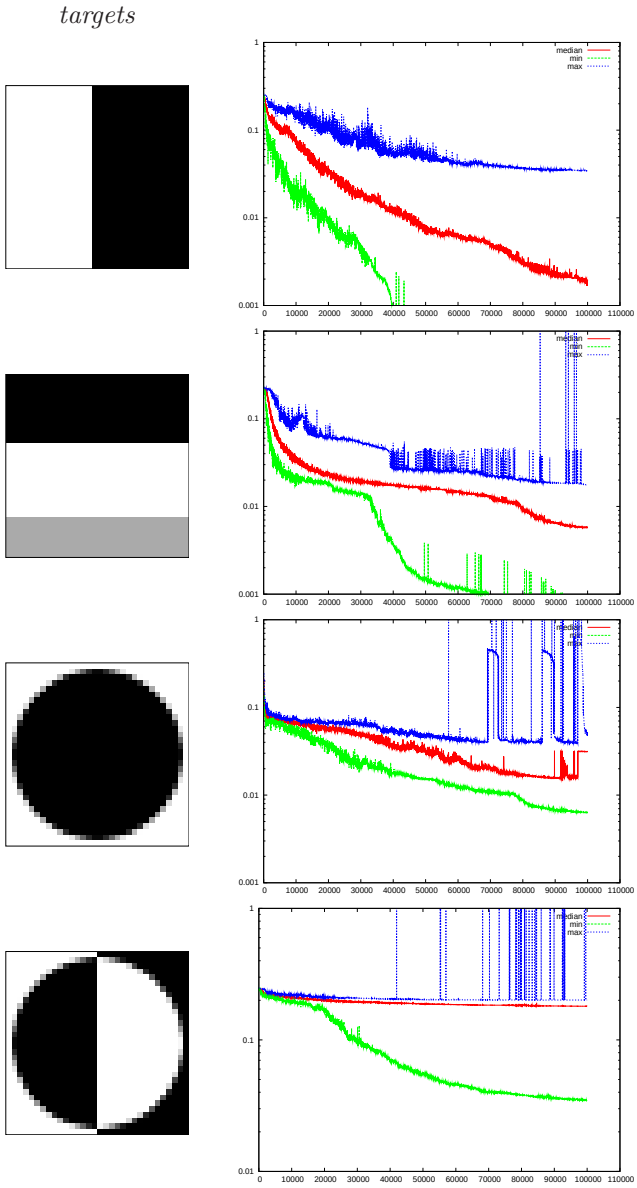


Figure 4.14: Diffusion based approach, median, minimum and maximum fitness over 32 runs. Vertical axis is the fitness in log scale, while horizontal axis is the number of evaluations.

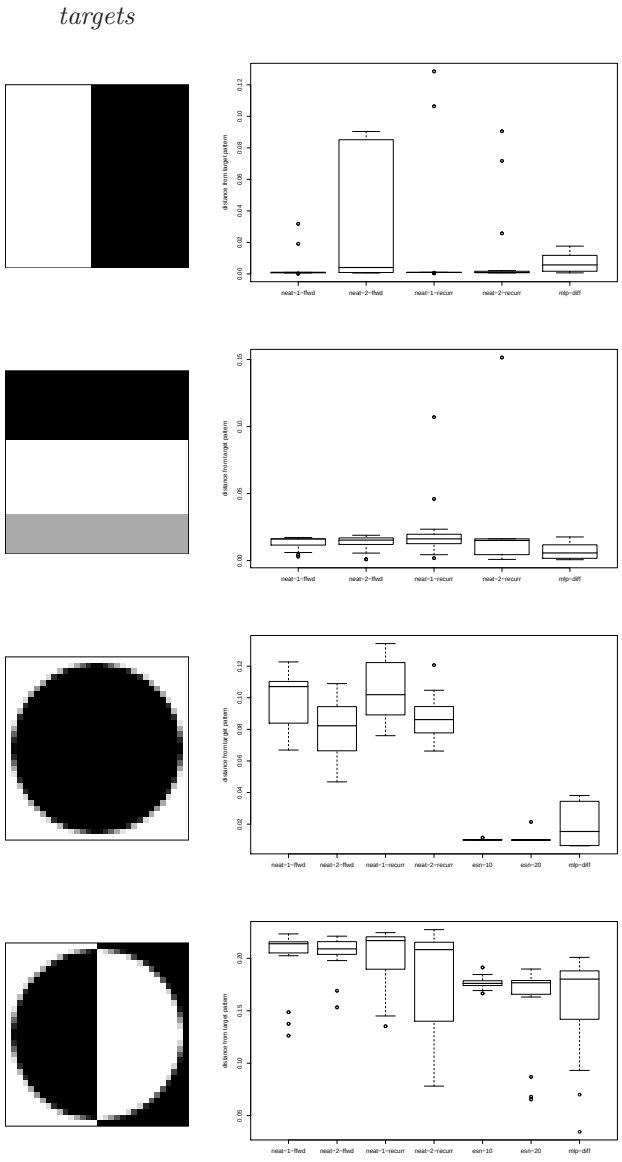


Figure 4.15: The boxplots show the distribution of the best pattern matching score obtained by our runs, for each pattern and each approaches. *neat-1-ffwd*, *neat-2-ffwd*, *neat-1-recurr* and *neat-2-recurr* stand for our experiments with NEAT and general neural networks, either feedforward or recurrent, with one or two chemicals. *esn-10* and *esn-20* stand for the ESN based approach, with reservoirs of respectively 10 and 20 neurons. *mlp-diff* stand for the diffusion based approach. The diffusion based approach is always doing as well or better than the other approaches studied here. Note that for the ESN based approach, only the *disc* and *half discs* have been studied.

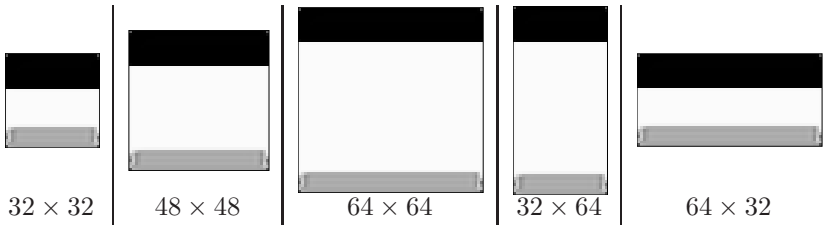


Figure 4.16: The same diffusion based controller, optimized to match the *three-bands* target, used in different grid sizes. While the *three-bands* target is a 32×32 pattern, the controller still features a stable attractor. Those attractors fit the three bands pattern, with the correct gray levels. However, the relative proportions of the bands are not the same.

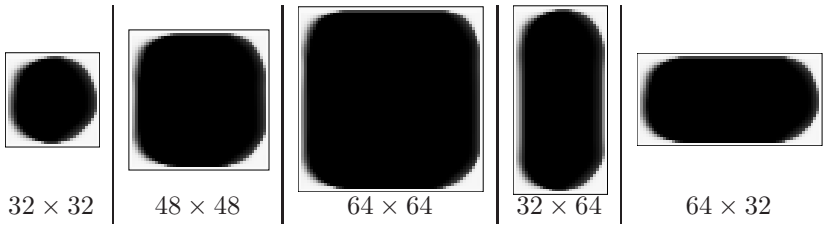


Figure 4.17: The same diffusion based controller, optimized to match the *disc* target, used in different grid sizes. While the *disc* target is a 32×32 pattern, the controller still features a stable attractor. Note the non-homothetic scaling of the pattern: the corners features keep their round shape, while the side features remain straight.

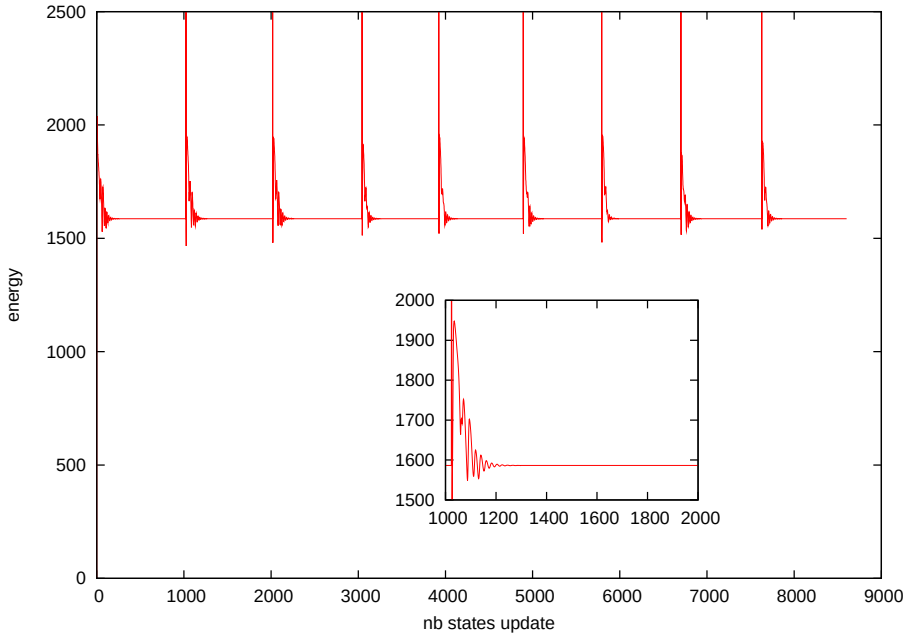


Figure 4.18: Energy of the cells of a 32×32 flag, controlled by the best controller found during one of our optimization runs. The target pattern is the *half-discs* pattern. Each time the energy variance over 16 time steps is 10^{-15}, a zero centered gaussian noise is added to each cells states for one time step, with deviation 1. This results in an energy spike. After a burst, the energy always returns to the same level, with a damped oscillations dynamic. Thus, the optimized controller seems to have a very strong attractor, coding for the target pattern.

cell states during the development just slows down the convergence to a set of states which express patterns close to the target pattern. If higher level of noise are employed (standard deviation of 1), once the noise addition is stopped, the flag restores the pattern it was optimised for. Though, such a treatment can change the dynamic of the flag. It can either converge to a new attractor (as shown on figure 4.19), or in some rare cases have a pseudo-periodic trajectory. The switching to a new attractor or a trajectory is triggered by a level noise highly individual dependent. This latter phenomenon was also found with the *NEAT* based development processes. Applying high noise levels to the cell states can then restore the original evolved attractor state, close to the target pattern.

Thus, a diffusion-based model features stable development, like both previous approaches. Also, reusing an optimized controller with diffusion-based model in different flags size does not break that stability property, as show in figure 4.16. Indeed, the grey levels and some features of the pattern are

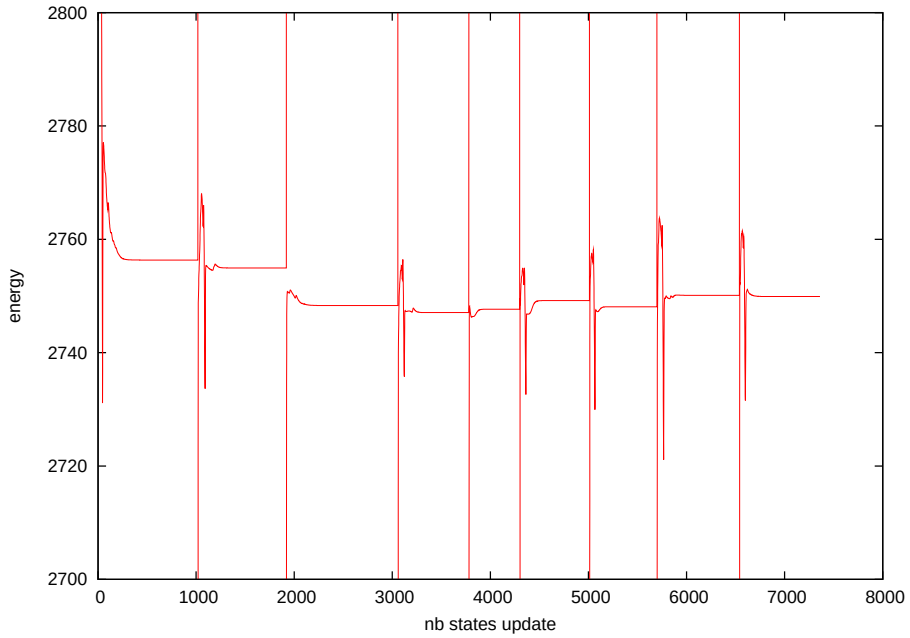


Figure 4.19: Energy of the cells of a 32×32 flag, controlled by the best controller found during one of our optimization runs. The target pattern is the *disc* pattern. Each time the energy variance over 16 time steps is bellow 10^{-15}, a zero centered gaussian noise is added to each cells states for one time step, with deviation 1. This results in an energy spike. After a burst, the energy does not always returns to the same level. Thus, the optimized controller have several attractors. In that particular example, each attractor codes for a different pattern, which slight variations of the target pattern : a disc. It's not always the case, a different stable energy level can code for a different pattern.

preserved (like the bands of the band patterns), even if the original relative proportions of the pattern features are lost. Again, this phenomenon has been noticed for all the optimized controllers. An homothetic scaling of the pattern, without doing further optimization, would need at least an adjustment of the diffusion process: a Gaussian blur of radius 2 for a flag stretched by a factor 2.

4.8 Development and stability

In our previous experiments, for the three approaches, the optimized controllers always code for development processes that end up to a steady state. Although those steady states are not always global attractors, they are at least local attractors. Since the controllers architecture are different, one might suspect that the stability property is independent of the control architecture. We hypothetize that the optimized development processes always end up to steady states because of the way we evaluate them. With the energy based stopping criterion, development processes that end up on a transient state are discarded.

This section will validate this hypothesis using a simple experimental setup. We will look at the influence of the number of development steps, and of the stopping criterion that is used during the evaluation of the candidate cell controllers. The observed effects are the similarity of the final expressed pattern with the target pattern, and the stability (ie the self-healing capacity) of the best solutions. Despite the strong impact of the development halting criterion, such a study had never done to the best of our knowledge. We carry out our experiments with the diffusion-based model for the embryogenic flag, described in previous Section 4.7.

4.8.1 Experimental setup

The developmental model is setup exactly as in the previous section: one internal chemical and two external chemicals. However, it is optimized using two different stopping criteria for the development when evaluating a candidate parameter setting. The first one, called *fixed*, consists simply in stopping the development after a fixed number of steps (here values 16, 32, 64, 128, 256, 512, and 1024 steps will be used). The other stopping criterion is the one that was introduced in the Section 4.4.2, whose goal is to enforce the emergence of a steady state. If the candidate controller does not satisfy this criterion after 1024 development steps, then it receives the worst possible fitness, 1. All settings are tested with 32 optimization runs, that are stopped after a maximum of 10^5 fitness evaluations.

4.8.2 Results and discussion

One striking result is, in the case of the *fixed* stopping criterion, the strong relationship between the development time and the median distance of the expressed pattern of the best controller to the target pattern. Shorter development

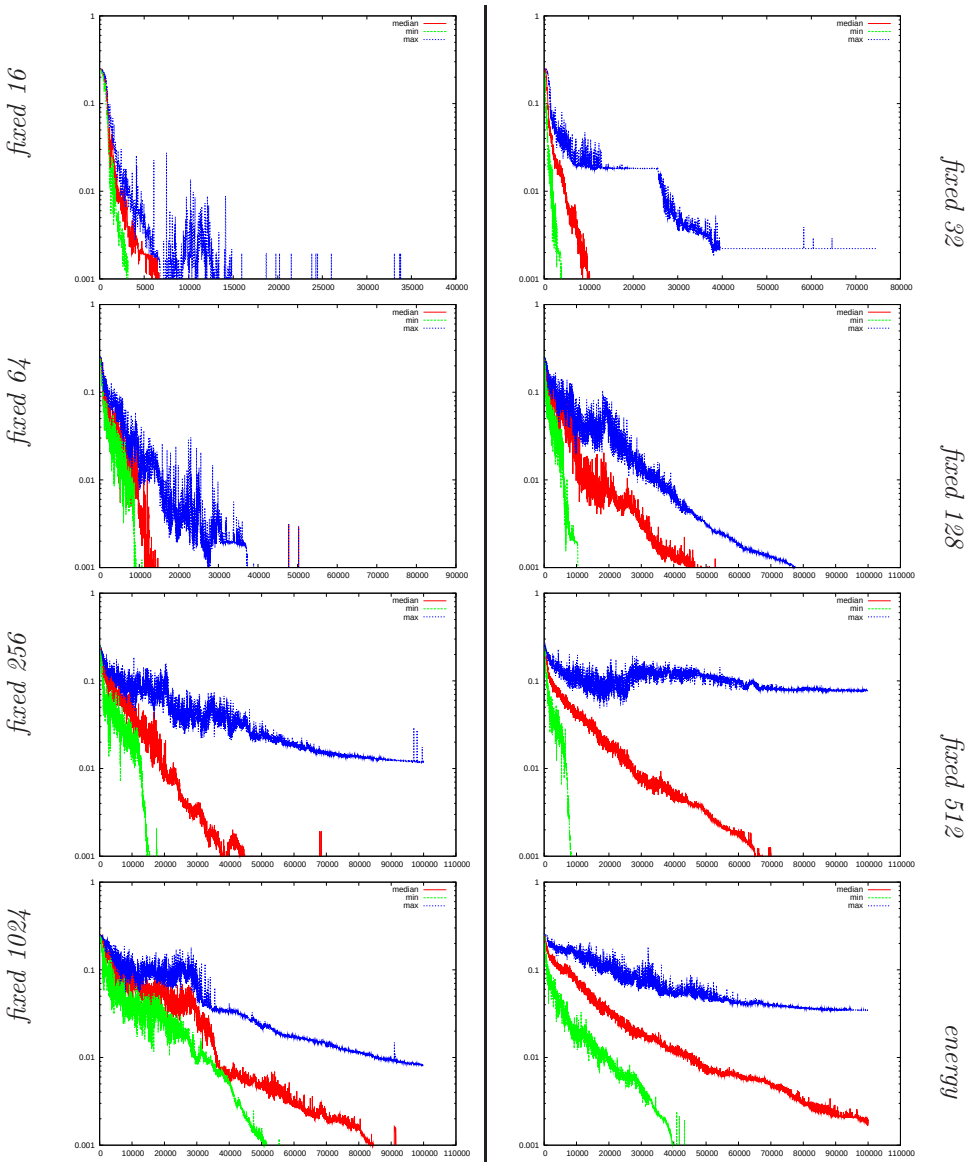


Figure 4.20: Median, minimum and maximum fitness over 32 runs, for the *two bands* pattern. Vertical axis is the fitness in log scale, while horizontal axis is the number of evaluations. The stopping criterion appears on the side of each plot.

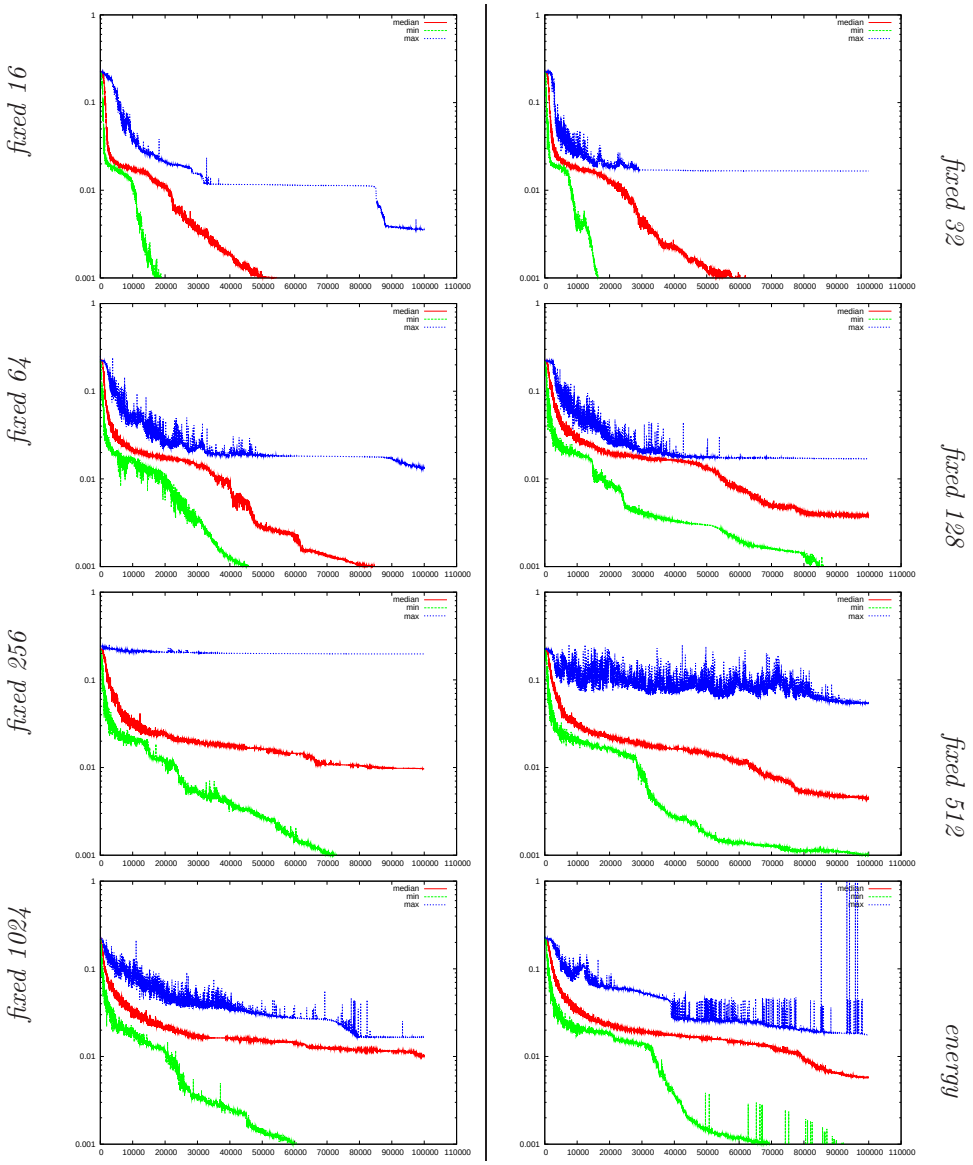


Figure 4.21: Median, minimum and maximum fitness over 32 runs, for the *three bands* pattern. Vertical axis is the fitness in log scale, while horizontal axis is the number of evaluations. The stopping criterion appears on the side of each plot.

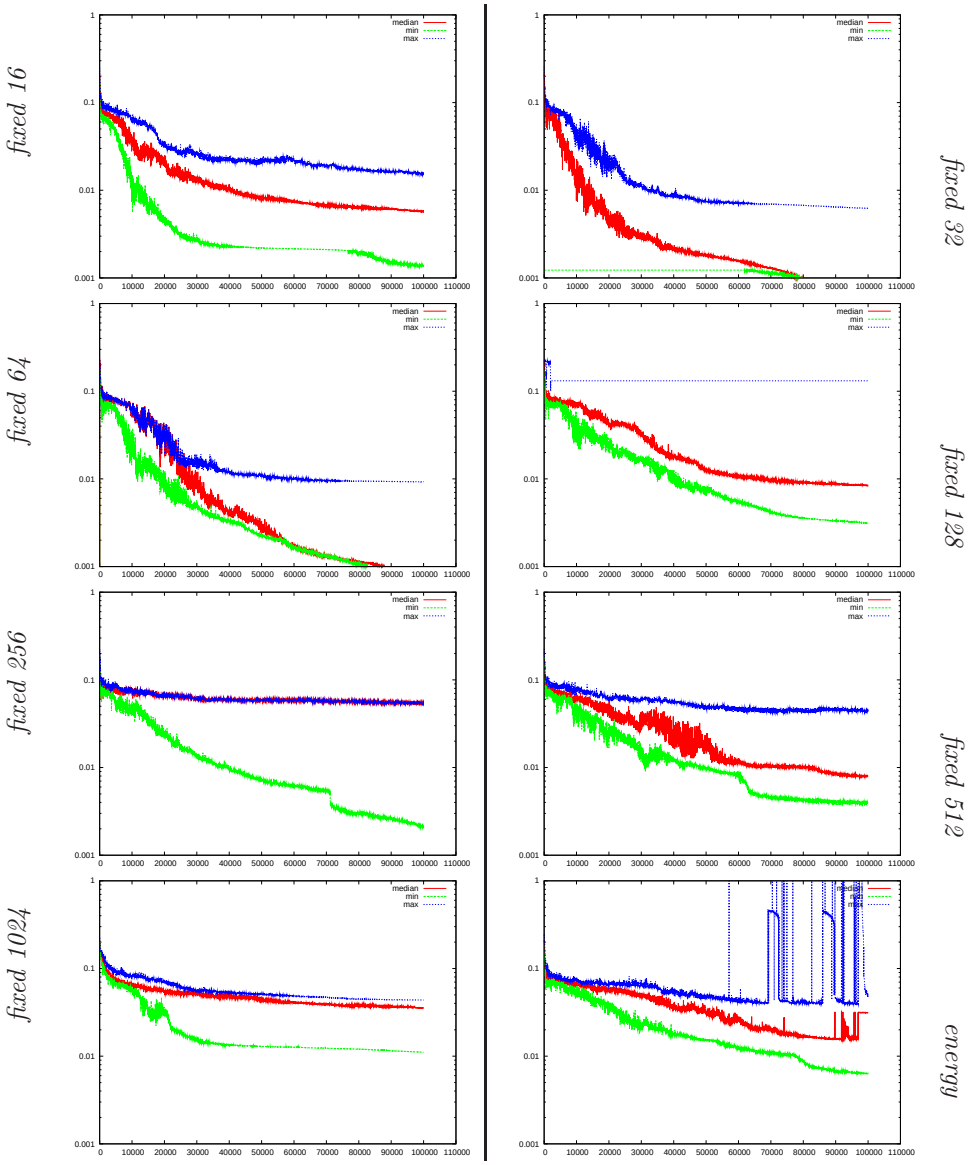


Figure 4.22: Median, minimum and maximum fitness over 32 runs, for the *disc* pattern. Vertical axis is the fitness in log scale, while horizontal axis is the number of evaluations. The stopping criterion appears on the side of each plot.

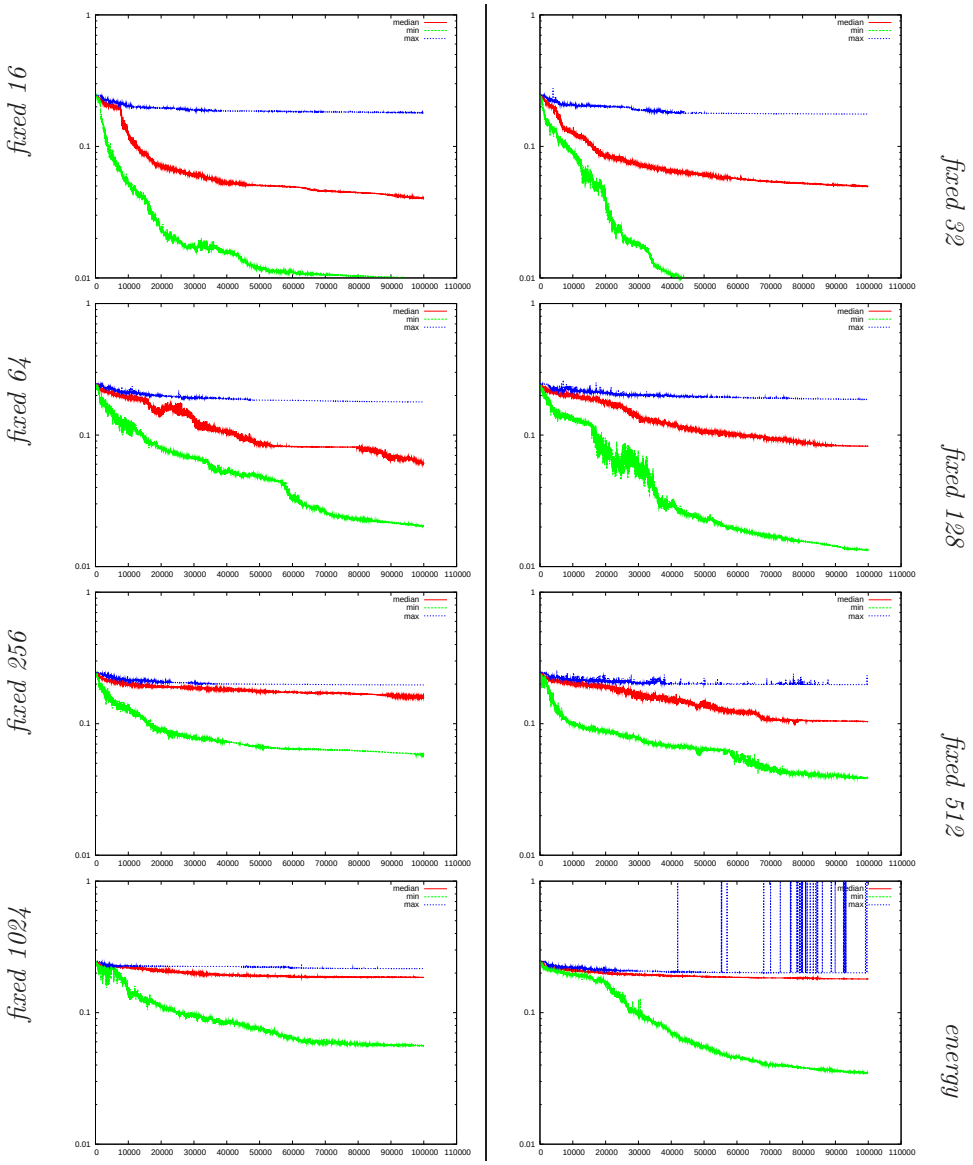


Figure 4.23: Median, minimum and maximum fitness over 32 runs, for the *half-discs* pattern. Vertical axis is the fitness in log scale, while horizontal axis is the number of evaluations. The stopping criterion appears on the side of each plot.

times clearly lead to better fitnesses. Indeed, a development of 16 steps is the minimum possible for a 32×32 flag, it is the time for chemical waves from the borders to meet at the center of the flag.

Furthermore, for all the runs that have been performed, the state reached by the flags optimized with the *fixed* criterion is never a stable state, but simply a point of the trajectory of the dynamical system: it is a transient state. figure 4.24 is a very representative example of such behavior. The trajectories in the flag state space are mostly pseudo-periodic, the remaining ones are either chaotic or perfectly periodic. By adding a Gaussian noise to the cells states during the development of those flags, one can change the pseudo-periodic or the periodic regime. The minimum standard deviation level of the noise needed to perform such a change is highly variable. The flags obtained from the runs with the higher development time have slow dynamics, and are less perturbed by low levels of noise (around 10^{-1}). However, they do not stabilize to a state that expresses the target pattern, if they stabilize at all.

In the case of the best controllers found by using the energy-based stopping criterion, the fitness scores achieved by the expressed pattern are better or equivalent to those reached by the *fixed* runs with 1024 steps per evaluation. For the sole optimality of the similarity to the target pattern, the *fixed* runs with low development times are clearly the best. Those runs reach better scores with fewer evaluations and fewer development steps, thus using much less computer resources. But the patterns that are reached are always transient states: the stability, the reusability, and the robustness are totally absent.

Changing the stopping criterion of the development process used during the evaluation of the candidate cell controllers completely changes the optimization problem. We were now asking for a development process that, at a given time, expresses a given pattern, and we indeed obtained what we asked for. The pattern is well approximated at precisely this number of steps, without any care about stability. It seems to be an easy problem, especially when that fixed number of steps is small. However, the state that is reached is not a stable state, destroying any chance of a reuse capacity. Controllers obtained that way are thus not good candidates to an expression of a *module*. On the opposite, by aiming at controllers that lead to a stable state, the likelihood to have a stable attractor state is much higher. Even if the optimization problem is harder, the solutions provided are much more consistent with the motivation underlying the cellular development based approaches: controllers that express very similar patterns when put in similar, but distinct, situations.

4.9 Conclusion

The embryogenic flags problem has allowed us to investigate many ideas on cellular representations within a common framework. From this point of view, we compared and discussed previous work. However, because of the numerous variations around the embryogenic flags, it is still quite hard to make accurate comparisons. For instance, from one approach to another, the pattern matching

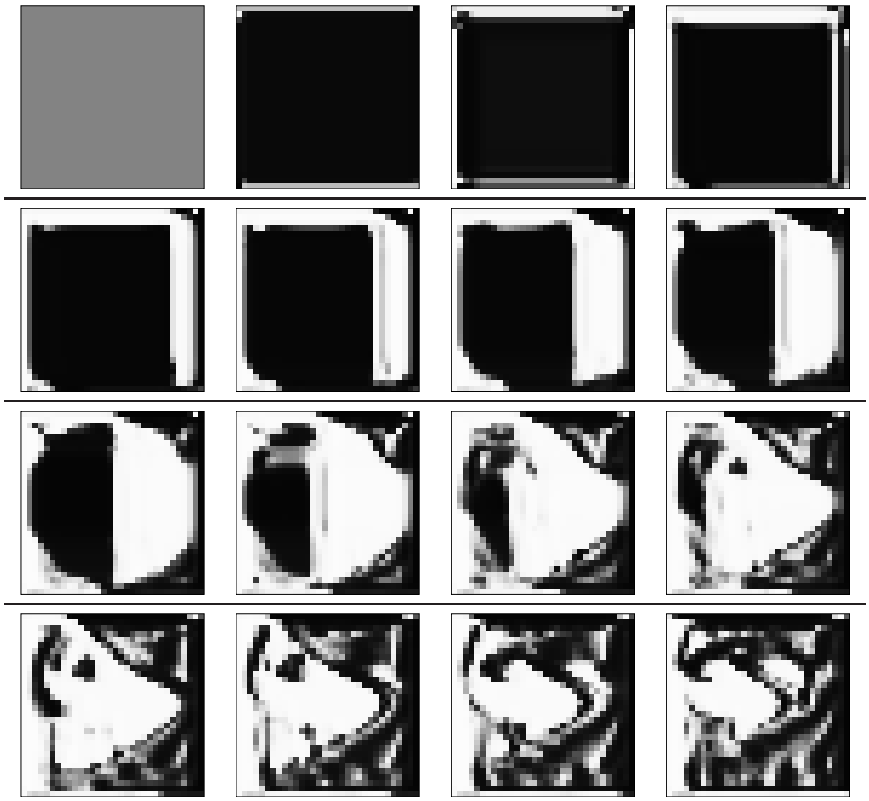


Figure 4.24: Sequence of the expressed pattern of a flag controller, optimized to match the *half discs* target. Only the even steps are shown, from left to right and top to bottom. The *fixed* stopping criterion was used during the evolution with 16 development steps. Note the really good matching at the step 16, and the unstable dynamic at further development steps.

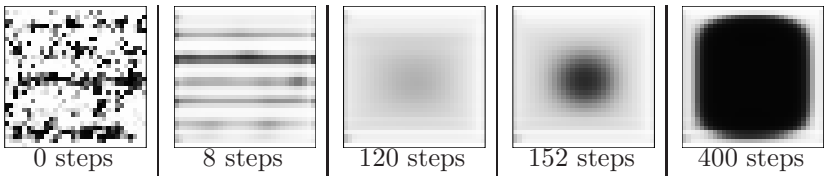


Figure 4.25: Sequence of the expressed pattern of a flag controller, optimized to match the *disc* target. The *fixed* energy criterion was employed for the evaluations. During 10 steps, a Gaussian noise of standard deviation 1.0 and zero mean was added to both chemicals and states of each cell. Then, the noise addition is stopped while the development continues. After the noise addition, the expressed pattern is completely destroyed. Soon after, the flag generates coherent waves. Then, about 100 steps later, the oscillations stops, and a single, slow, pulse appears, generating the disc pattern.

measure is never the same, nor the optimization method. Here, in the context of Evolutionary Design, our main objectives is a streamlined developmental model, that provides evolvable and stable development processes, with meaningful parameters.

One of the main contributions of our experiments, is the stopping criterion for the development process. It is shown to be a critical issue with respect to obtaining stable development processes. We showed that restricting our goal to finding steady-states of the development process, rather than considering any possible development process, does make the optimization problem harder, though still tractable. With a simple heuristic, waiting for a global energy measure to stall, we have been able to find such stable development processes without the need to evaluate a candidate development process over several initial conditions. This is indeed, the best solutions we found for the pattern matching task were quite resilient to perturbations during development, featuring levels of self-healing yet unseen in previous works. Further more, the heuristic we proposed is quite general, simple, and not limited to the embryogenic flag problem. Nevertheless, we had to penalize in a very harsh manner the candidate controllers that lead to too long developmental sequences. A more subtle penalization scheme might help to improve the expressed patterns scores.

In our efforts to obtain a streamlined developmental model, we combined several strategies borrowed from previous works. The first one was to rely on continuous states: indeed, when considering our longer term goal, Evolutionary Design, this seemed mandatory, as the information gathered from physical simulations is continuous, as well as most of the physics modelisation. Another idea was to completely remove any cell duplication mechanism: the cells populate the whole space from the beginning. They exchange information, compute over space and time, change their physical environment, and read from it. We believe that, compared to models that include growth and death processes, it reduces the complexity of the developmental model, making it more homogeneous, even though it departs from the biological paradigm.

Our first model used a generic recurrent neural network for the update and expression functions. This approach was based on the *NEAT* algorithm, and provided us with our first interesting results on stable development processes. But *NEAT* has a lot of parameters to tune, and thus did not really fit our goal of streamlining. We were able to switch to a fully parametric approach, where we do not need to evolve the topology of the neural network, relying on random connectivity, using Echo State Networks. Those networks allowed us to use a purely parametric and almost parameter free optimization to optimize the development processes. This approach gave similar performances than the best results of the *NEAT*-based approach, while retaining the ability to obtain stable development processes.

Then, we broke down in several sub-parts the single recurrent neural network we were using to encode development processes. We turned the cells communication mechanism into an had-hoc diffusion process, as introduced in Miller's [85, 86] and Federici's [106] works. That way, we obtained the same pattern matching scores than those obtained with the *NEAT* based approach, but with

much less states per cells and a lot less fitness evaluations.

The diffusion-based model is close to a discretised reaction-diffusion system. Thus, we might obtain further improvements by making the developmental model even closer to a reaction-diffusion system. One interesting side effect of using a diffusion process is that such a process is defined independently from the topology of the cells environment. Diffusion processes can be defined for any regular lattice, but also on irregular tilings of spaces of any dimensions. So a fully diffusion-based developmental approach, as reaction-diffusion systems are, would be versatile, relieving us from constraint of using a regular square lattice. Since most physical simulations rely on irregular tilings (e.g. in Finite Elements Methods), we believe that such a feature is mandatory.

The embryogenic flag is a simple testbed, that has already been used to introduce and test ideas about cellular development processes. Comparing the state of the art work in order to try to derive new ideas is difficult. The pattern matching problem might need to be extended in order to study the scalability and the self-healing properties of the embryogenic flags. Patterns defined independently from the size of the cell grid go in this direction. Simple vector graphics are one example. Interesting patterns could be statistically defined patterns, like *"a white unit square packed with as much as possible black blobs of area 0.05"*. Irregular tilings would be also interesting to study cellular approaches that do not rely too much on particularities of a given neighbourhood.

Also, cellular approaches are often intended to be used in an Evolutionary Design system. The embryogenic flag problem features no coupling with any physical phenomenon, questioning its relevance to Evolutionary Design, especially when referring to Estevez's work [29]. His framework is very similar to that of the embryogenic flags. It also features simplified yet complex physics and has a well-defined, non ambiguous fitness function. Results demonstrate some scalability and self-healing properties. We believe that such problems can pave the way to competent cellular approaches for real world design problems. As a first step toward this direction, in next chapter 5, we present some truss structures design problem addressed with a cellular representation that is aware of the relevant mechanical quantities.

We were in the jungle, there were too many of us, we had access to too much money, too much equipment, and little by little, we went insane.

Francis Ford Coppola

5

A cellular development model for structural truss design

This chapter describes an Evolutionary Design system based on a cellular development model for structural truss design. We first introduce structural trusses and the related design problems, and previous attempts to solve them by means of Evolutionary Design. We describe with extensive details *EiffelBlob*, a cellular model that is able to morph trusses using distributed, local physical feedback. Then, experimental results of an early implementation of this model, on the classical cantilever benchmark, are presented and discussed. Finally, we discuss the contributions brought by *EiffelBlob* and the ways to make it ready for more complex problems.

5.1 Structural trusses

Truss structures are ubiquitously used in architecture and engineering, both in real-world applications and for prototyping purposes. Although simple to represent and evaluate, they can be very complex, and are still a challenge for optimum design. The main goal when designing a truss structure is to make it as light as possible while being strong enough to withstand a given situation, i.e. exhibiting as small deformations and stress as possible when extreme loads are applied. Solving the inverse problem (what is the best structure for a given loading?) relies on being able to solve the direct problem (what are the deformations and stresses of a structure for a known loading?).

Trusses are made of beams connected by joints. There as several possible models for a beam, both in 2 and 3 dimensions, depending on the mechanical phenomena that are taken into account in the modeling and on the physical properties of the materials of the beams. In this work, the simplest 2d bar model with linear material will be considered: the effects of flexion are ignored, a beam can transmit forces only by compression or elongation, and the stress is proportional to the strain. Moreover, theam weight of the structure is neglected. A beam is hence modeled as a rigid spring, whose stiffness is the product of its cross-section area A and the Young modulus E of the material it is made of. In

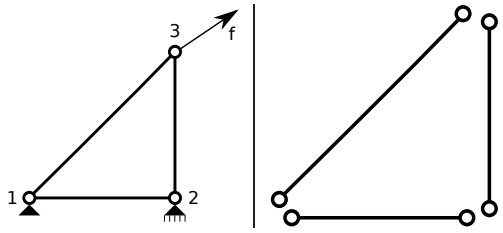


Figure 5.1: On the left, a very simple structure. The joint 2 is completly fixed, the joint 1 can move only on the horizontal axis, and joint 3 is completly free. A force is applied to the joint 3. In a perfectly static structure, the sum of forces at the joints is zero. This equilibrium can be written as a system of linear equations, which solution are the *virtual displacements* of the joints. From the *virtual displacements* we can deduce the stress on each beam.

this planar model, each joint has at most 2 degrees of freedom.

From there on, solving the direct problem using a Finite Element Method (FEM) is straightforward: the unknown variables are the displacements of all joints, the sum of all forces applied to a given joint is 0 at equilibrium, and each beam results in a linear equation involving the displacements of its 2 joints and the forces applied there.

To each joints, we associate a load: a force f applied to it. We neglect the effect of the structure weight, which is the practice in structural engineering. The goal of a structural analysis is to determine how much compressed or stretched is each beam when the truss is at equilibrium: determine the strain of all beams. For different materials, the practical tolerable strains are known. Because each beam can be either compressed or stretched, each joint can move in the 2d space: this move is called virtual displacement, denoted u. The 2 joints of indexes i and j displacements and loads of a single beam are linked by the following relation:

$$\begin{bmatrix} f_{x_i} \\ f_{y_i} \\ f_{x_j} \\ f_{y_j} \end{bmatrix} = \frac{AE}{L} \begin{bmatrix} c^2 & sc & -c^2 & -sc \\ sc & s^2 & -sc & -s^2 \\ -c^2 & -sc & c^2 & sc \\ -sc & -s^2 & sc & s^2 \end{bmatrix} \begin{bmatrix} u_{x_i} \\ u_{y_i} \\ u_{x_j} \\ u_{y_j} \end{bmatrix} \quad \begin{matrix} s = \sin\beta \\ c = \cos\beta \end{matrix}$$

where L is the length of the beam and β it's angle in the global coordinate system. The matrix associated to each beam is called *local stiffness matrix*. Those matrices are then *assembled* to form the *stiffness matrix* K, summarizing the whole truss equilibrium. K is positive-definite, symmetric and usually very sparse, and hence solving $KU = F$ is straightforward (F contains the load for each joint), even for large sizes of K. From the joints displacements U, one can deduce the beams strains and stresses.

5.1.1 Optimization of Truss Structures

While the direct problem can be considered solved by FEMs, there is not yet a general consensus on how to solve the inverse problem, and optimize a truss structure for a given situation. There are three kinds of truss design problems. The *sizing problem* tunes the cross-sections of beams on a fixed truss geometry. The *configuration optimization* additionaly moves the positions of the joints (and hence the beam length) for a fixed topology (i.e. the number of trusses and their connections are fixed). Both problems deal with a fixed number of design variables, generaly continuous (though the cross-sections can be restricted to practically available discrete values). The *topology optimization* consists in finding an optimal connectivity of the beams, and how to represent a general truss structure remains an open issue.

With their ability to cope with any representations without making strong assumptions about the fitness function, evolutionary algorithms has been popular for truss structure design since their early years, as recently surveyed by R. Kicinger et al. [65]. Most of these early works focus on a direct representation of the truss structure, where the genotype is a list of design variables. The optimization then consists of moving slightly the joints positions and/or changing the beams sections. Another direct approach, as in [22], is to start from a dense truss (called *ground structure*), and let the optimization algorithm remove most of the beams. In this case, the representation is the connectivity matrix. More recently, other approaches consisted in optimizing a transformation of an initial truss by a function rather than directly the truss itself. In [123], the authors represent the transformation as a program, optimized by Linear Genetic Programming. More recent works are amenable to Artificial Embryogeny, and have been presented in detail in chapter 2 Section 2.3.

5.2 EiffelBlob

We now introduce *EiffelBlob*, an Evolutionary Design system based on a cellular development model for structural trusses, that builds on the ideas developed in previous chapter and adds some feedback from the physical environement. Candidate trusses are represented as an initial truss morphed into a new truss by a transformation. While the phenotypes are trusses, the genotypes are a real valued vectors, the parameters, that control truss transformations. The evolutionary part is hence straightforward, whereas the originality of *EiffelBlob* lies in the cellular model: The truss transformation process is done by cells that communicate between themselves, and have access to a local, problem domain specific feedback from the mechanical behavior of the transformed truss structure.

The aim of the development model is to describe smooth transforms of trusses. Thus, a search in the parameters space of our model is equivalent to a walk in a space of truss transformations. We are searching for the transformation that takes an initial truss and turns it into an optimal truss according to

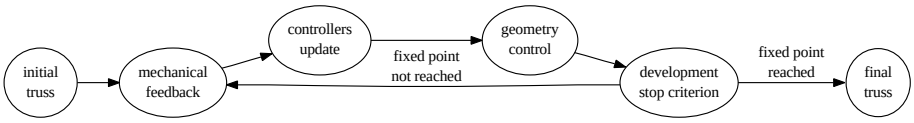


Figure 5.2: The steps of the development process

the problem at hand. An auxilliary goal is to evolve a generalized solution, i.e. a transformation that also work well when applied to trusses that are close to the one used during the search. In order to achieve this, the development model is coupled to a structural analysis algorithm in a stigmergic fashion: the truss transformation is performed by iterative steps, each step modifies the truss, and depends on the current truss state (i.e. its mechanical properties). At some point, the transformation steps will eventually stop modifying the structure, thus ending the transformation process.

Two dimensional trusses are planar graphs, if we consider their joints as the vertices and the beams as the edges. Let us consider the dual graph of this truss: the edges of the dual graphs are the *alveoli* of the truss. Beams, alveoli and joints are considered in *EiffelBlob* as *cells*, i.e. entities having both an internal state (a vector of real values, termed *chemicals*) and an update rule, or controller. The controller, implemented here as a Multi-Layer Perceptron (MLP) with fixed topology, takes inputs from the internal states of the cell and all its neighbors, and from the outside world, the results of the FEM simulation. Its outputs are some updates of its internal states, and some commands that will trigger physical alterations of the truss elements it controls. New external inputs are then gathered, and the process iterates until a fixed-point is reached. The final state of the truss defines the phenotype corresponding to the genotype (the weights of the MLP), and its mechanical properties determine the fitness of this genotype. The complete truss is called the *organism*, and can be seen as a graph with three kinds of cells: the joint cells, the beam cells and the triangles (alveoli) cells. All beams (resp. joints, alveoli) cells share the same controller, while having their own internal state. Hence, the development of the organism is completely determined by the initial truss and the 3 MLPs, one controller for each type of cell. In particular, the complexity of the genotype does not depend on the complexity of the initial truss. With such a representation, truss design problems with ten or thousands beams are potentially tractable by an evolutionary algorithm, being only limited by the computational cost of the FEM analyzes.

5.2.1 Control Topologies

The trusses amenable to optimization by *EiffelBlob* must be embeddable on a triangular lattice: this is a restrictrion of general planar trusses. This design decision is indeed motivated by conservatism with respect to our previous work on the *embryogenic flags* described in Chapter 4. The *embryogenic flags* relies on

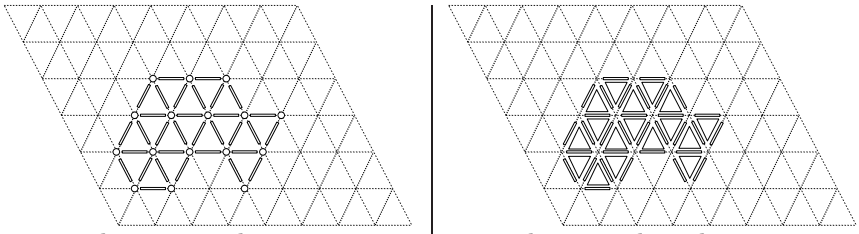


Figure 5.3: Two control architectures for the same truss (below). The top plot is a possible transformed truss.

square lattices with a Von Neumann or Moore neighborhood, each cell had exactly 4 neighbours. Here, we switched to a triangular lattice. As a consequence, joint cells have at most 6 neighbour beam cells, beam cells have exactly 2 joints neighbour cells. The alveoli are all triangles. Triangle cells have 3 neighbours beam cells, beam cells have at most 2 neighbours triangle cells. All cells hence have the same number of neighbours, and we avoid the problem of dealing with variable neighbourhoods, as would be the case for general planar trusses. Also, regular lattices might be easier to analyse compared to general graphs.

Beam cells act both as sensors and as effectors of the truss organisms since they get some feedback from the environment through accessing information about the beam strain, and trigger local alterations of the truss: increase or decrease of both the length and the cross-section of the beam, as will be detailled in forthcoming Section 5.2.3. We wanted to provide the beam cells with a way to diffuse information, and faced two possible choices: communication through the triangles, or through the joints, which we named respectively *edge-triangle* and *edge-joint* architecture. Figure 5.3 shows the same physical truss (top) with its corresponding two possible control architectures (bottom).

5.2.2 Control and information diffusion

Practically, controllers considered here are simple 3-layer MLPs with a fixed architecture (i.e. one single hidden layer). All beam (resp. joint, triangles) perceptrons have as many inputs as number of neighbour cells times number of state variables. Thus, if the state vectors S of the cells are of size n, beams perceptrons have $2n$ inputs, joints perceptrons have $6n$ inputs, and triangle perceptrons have $3n$ inputs. All beams (resp. joints, triangles) perceptrons

output a vector Δ_S of size n, and the state vectors S are updated as follows:

$$S(t+1) = (1-\alpha)S(t) + \alpha\Delta_{S(t)}$$

Parameter α allows us to tune the inertia of the internal states vectors S, and was set to 0.05 after few exploratory experiments. Beam cells have the capacity to change the beam size and section, which is the purpose of their special outputs Δ_L and Δ_R. Let be $L(t)$ and $R(t)$ the length and the radius of a beam at the developement step t, then they are updated as follows:

$$\begin{aligned} L(t+1) &= (1.0 + \Delta_L(t))L(t) \\ R(t+1) &= (1.0 + \Delta_R(t))R(t) \end{aligned}$$

This update scheme allows us to adapt the changes according to the scale of the length (respectively the radius) of a beam: small beams will have little changes, larger beams will have larger changes.

Neurons from the hidden layer and the cell state update output use the classical *tanh* activation function. However, the Δ_L and Δ_R outputs of the beam controller use a less common activation function:

$$\sigma(x) = \frac{2\delta}{e^{\frac{-1}{2}}\sqrt{2}} x e^{-x^2}$$

This function ensures a smooth modification, even with completely random controllers. The δ parameter is the maximum absolute value of $\sigma(x)$ and was fixed to 0.1 after few exploratory experiments.

5.2.3 Geometry control

At each iteration of the development, the controller for each beam provides values for Δ_L and Δ_R, the geometrical commands for the underlying beam. This triggers a problem: the trusses joints positions should be moved to satisfy the new beam lengths, but all moves might not be geometrically feasible (e.g. the triangular inequality could be violated).

In order to move the joint positions properly, the truss is considered as a mass-spring system where each joint has unit mass and each beam is a spring with unit stiffness and rest length equal to the virtual length provided after variation due to the controllers. This mass-spring system is then relaxed until it reaches an equilibrium state, which is performed using a damped Verlet integration scheme, as in [58], with an integration step $\Delta t = 10^{-2}$ and a damping factor $\beta = 10^{-2}$.

$$\vec{x}(t+\Delta t) = (2-\beta)\vec{x}(t) - (1-\beta)\vec{x}(t-\Delta t) + \vec{a}(t)\Delta t^2$$

$\vec{x}(t)$ is the position of a node at time t and $\vec{a}(t)$ is the sum of all spring forces applied to this node. The computational cost of this integration scheme is close to that of the explicit Euler scheme, albeit with a greater stability. Another great strength of the Verlet integrator is its practical handling of geometrical

constraints: fixing the positions of some of the joints, or putting sliding conditions, is straighfoward. It can handle contacts too, so the programmer can define some areas where no joints are allowed. This way, many practical engineering constraints can be handled without difficulties: the structure should not be under water, should be plugged on a external structure, etc... Note that this procedure produces an implicit but global feedback to the cell controllers from the environment geometry, without any additional input.

The mass-spring system is relaxed until the variance of the kinetic energy of the masses, computed over 11 relaxation steps, is below a given threshold (a default value of 10^{-15}, close to the 64 bits floating point accuracy, was used here). Such a stopping criterion is immune to an early stop into a configuration which would not satisfy the controllers commands. Because of the non-zero damping factor, this always occurs in a finite (and small) number of relaxation steps, though, obviously, larger systems need longer relaxation time. While this relaxation process is time-consuming, it is relevant as it allows the programmer to freely transform the truss geometry, as the typical constraints of engineering problems are taken into account during the mass-spring relaxation phase.

5.2.4 Mechanical feedback

Right after the geometry update, a full FEM analysis is carried out, as described in Section 5.1. The corresponding linear system is solved by a standard iterative solver: a Conjugate Gradient algorithm with Jacobi preconditioning. Iterations are stopped whenever the residual norm is below 10^{-6}. Even though the algorithm theoreticaly converges in at most N_M steps (with N_M is the size of the matrix), it might in practice take much longer, depending on the condition number of the linear system. Hence an absolute limit on the number of iterations is also set ($20 \times N_M$), and the truss is considered invalid if the residual norm is still too high after that many iterations. In this latter case, development stops and a strongly penalized fitness value is returned for this truss. On the other hand, in the standard case, strain values obtained after convergence are used as inputs of the beam controllers, and development goes on.

5.2.5 Initialization and halting

The initial condition of the developmental process considers that all chemicals for communication between cells are set to zero. The initial geometry is user-defined, depending on the structural engineering problem at hand. It requires some little expertise, as initial experiments have demonstrated that chosing as a starting point a rough solution of the problem at hand, even though very suboptimal, represents an efficient bootstrapping for the optimization process. The geometric constraints for the joints and the loadings are also derived from the problem at hand.

The stopping criterion is implemented by monitoring a measure of the variance, over a sliding time window, of the global energy of the organism, computed as the L2 norm of the array of cell state vectors, as designed for the embryogenic

flag problem in Section 4.4.2. The development stops whenever the variance of this energy is smaller than 10^{-15} in any window of 16 consecutive time steps: this a fixed-point detection heuristic. However, penalization of trusses that fail to reach a fixed point is done here in a two-steps process: First a penalty is added to the fitness of the individuals whose development did not stop according to the energy criterion within a user-defined total number of iterations, but development nevertheless proceeds; Then, the individuals whose development did not stop after twice this number of iterations get an even poorer fitness, without any further Finite Element Analysis. Otherwise, the problem dependent fitness is computed on the truss produced by the development.

5.3 Experimentations

The *EiffelBlob* cellular model can morph a truss to an other. The transformation is tunable, and thanks to a mechanical feedback, have an accute domain knowledge for mechanical problems. An other effect of this feedback is the excepted capacity to morph similar trusses to a truss that is optimal according to a given objective. We did the following experiments in order to check the following properties:

- The capacity, for a state of the art search algorithm, to find a transformation that morph a initial sub-optimal truss into a better truss according to a given fitness measure.

- Once such a transformation found, the capacity of the resulting transformation to also improve, with respect to the same criterion, trusses that are similar (but not equal) to the one used during the optimization phase.

Thus, while the long term objective is to have a structural truss design framework that is suitable for a large scope of problems, we just want here to validate the *EiffelBlob* approach. A classical, well know structural truss problem and testbed is the cantilever problem. Cantilevers are one of those modest ubiquitous structures that literally are used as support everywhere. Here, the problem is defined as follows: two joints of the truss, on a same vertical axis, remains fixed. Those are the support joints. The joint with the lowest position is fully fixed, while the other can move along the vertical axis. A third joint of the structure is two meters right of the support joints. In the mass-spring simulation, its position remains fixed, but is it has two degrees of freedom and it is loaded with a vertical downward force, from the mechanical point of view. By changing the joints positions and the beams sections, we wish to have the lightest possible truss which respects to a constraint on the maximum strain on all beams.

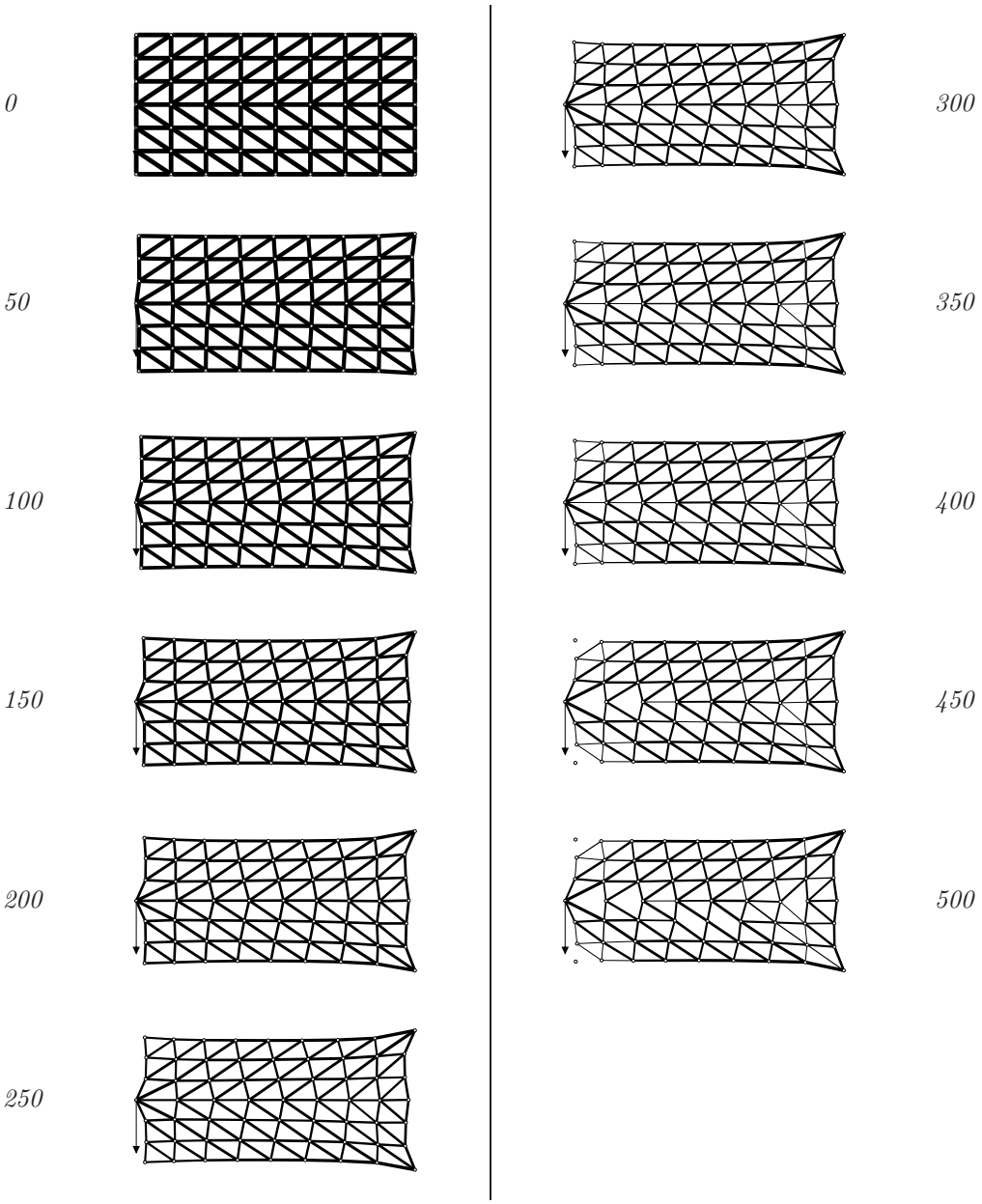
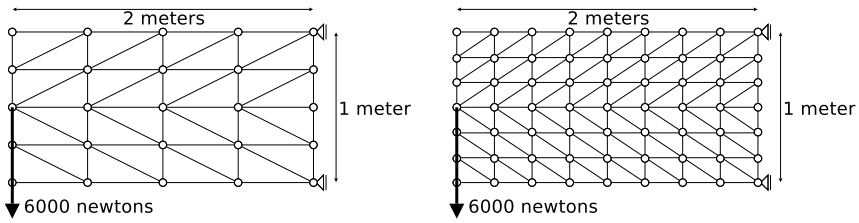


Figure 5.4: An example of a developmental sequence with *EiffelBlob* for the cantilever problem. The number of developments are on the side. The last stage is the fixed-point. The thickness of each line is proportional to the radius of the corresponding beam.



4×4 cantilever *8×6* cantilever

Figure 5.5: The two initial trusses used for the developemental sequences.

5.3.1 Optimization of a developement process

Initial trusses for the developement

In order to solve the cantilever design problem with *EiffelBlob* we need to define the initial truss used for the development phase of the evaluation function, and the fitness of candidate trusses. The initial trusses are the *4×4* and *8×6* cantilevers shown on figure 5.5. Note that while having differents numbers of beams and joints, the genotype size remains the same: the weights of the MLP. We used a beam material with the characteristics of a steel commonly employed in buildings: Yound modulus of 210×10^9 and density of 7860.0.

Control Architecture and Controllers

We tested both the *edge-triangle* and *edge-joint* architectures (see Section 5.2.1 and figure 5.3). The *edge-joint* architecture might seem more natural for somebody with a background in structural mechanic. In contrast, the *edge-triangle* architecture is less intuitive but is slightly more efficient in term of controller complexity (38 versus 41 parameters). The controllers, as explained previously, are perceptrons with a single hidden layer. Only 4 hidden neurons were sufficient here for the joints cells, and a single hidden neuron for the beam (respectively triangle) cells in the case of the *edge-joint* (respectively *edge-triangle*) architecture. The cell state vectors only have one dimension: the state of a cell is described by a single scalar value. Altogether, the *edge-joint* cellular development approach requires the optimization of 41 real parameters, while the *edge-triangle* cellular development approach has 38 real parameters.

Evaluation Function

The fitness function is the weight of the truss structure, to be minimized. So, according to the description of *EiffelBlob* the evaluation function is:

1. Take the initial truss.

2. Setup the cells in their initial states, the perceptron weights are taken from the genotype.

3. Apply development steps until fixed point is reached.

4. Return the weight of the obtained truss

However, we also want all beams to stay in the secure regime as far as strain is concerned: If the development leads to a truss with at least one beam strain over 0.1, the individual receives a very bad fitness: the weight W_{init} of the truss used for the developement initialization.

Furthermore, as described earlier, individuals whose the developement process last too long are penalized: after 2048 steps, a penalty is added to the fitness as follows, where S is the number of developement steps:

$$P = \frac{W_{init}}{2048^2}(S - 2048)^2$$

With this penalty, an individual that needs more than 4096 development steps will have a fitness greater than W_{init}, hence the development is stopped. Finally, the fitness is normalized by dividing the penalised weight by the weight of the initial truss.

Optimizer Setup

In order to optimize the weights of the MLP controllers, the *CMAES with restart* algorithm [3] has been used (using the available C implementation by the original authors). All *CMAES* parameters and heuristics were set to their default values, excepted of initial mutation step size, set to 10^{-2}. All experiments are run for 10000 evaluations, and results are statistics over 20 runs.

Results

Statistical data of the optimization runs are summarized in figures 5.6, 5.7, 5.8 and 5.9. The *CMAES with restart* algorithm is not an elitist algorithm, and this explains the jumps of the mean minimum weight curves, that correspond to restarts of *CMAES*. Another phenomenon contributed to those jumps. The mass-spring system approach used for the geometry control does not handle well situations where close-to-flat triangles appear, leading to singularities and pathologic geometries. Such geometries then pollute the FEA routine, then the controllers, and as a result the developement process degenerate.

The *edge-joint* and the *edge-triangle* architectures distribute the information in different ways, one way could be better than the other one. When observing the best fitness obtained, we could not observe any statistically significant difference, with a Student's t-test, between the two architectures. We should use much harder problems, though it could be that both approaches are in fact equivalent.

5.3.2 Comparison with the Direct Representation

We compared the *EiffelBlob* representation with a direct representation of trusses. In the direct representation, the genotype is a vector with all coordinates of the

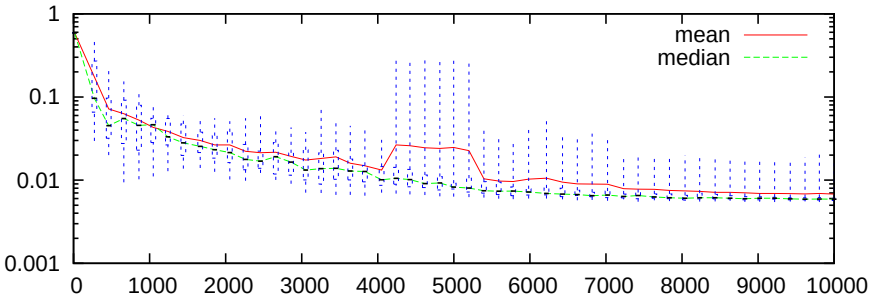


Figure 5.6: *4×4* cantilever optimization with *EiffelBlob edge-joint* architecture. X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

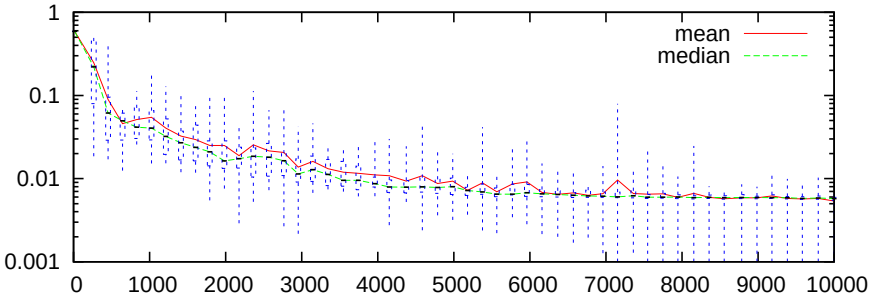


Figure 5.7: *4×4* cantilever optimization with *EiffelBlob edge-triangle* architecture . X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

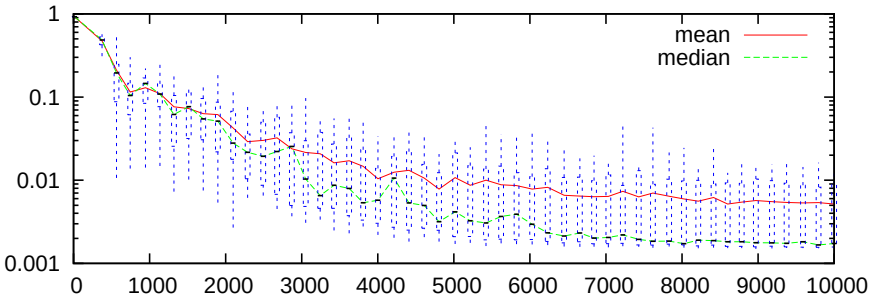


Figure 5.8: *8×6* cantilever optimization with *EiffelBlob edge-joint* architecture. X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

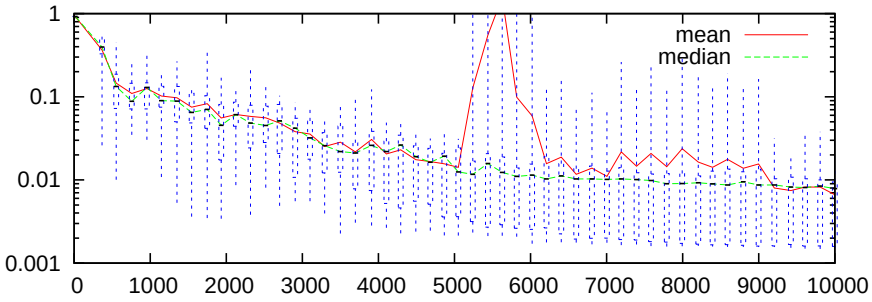


Figure 5.9: *8×6* cantilever optimization with *EiffelBlob* *edge-triangle* architecture. X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

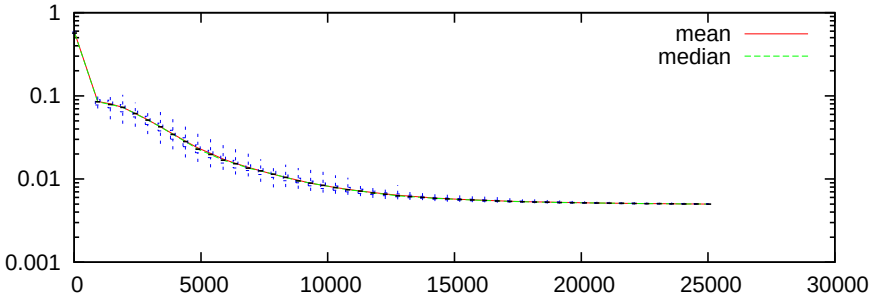


Figure 5.10: *4×4* cantilever optimization with direct representation. X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

joints, plus all sections of the beams. The same optimization algorithm (CMA-ES) with the same settings is used. The initial genotype corresponds to the initial trusses employed to boostrap the development with *EiffelBlob*In the case of *4×4* cantilever, the genotype is a vector of size 106, while it is of size 284 for the *8×6* cantilever. All experiments are run for at least 25000 evaluations, and results are statistics over 100 runs.

The reader should keep in mind these are two completly different problems. With *EiffelBlob*, we optimize truss transformations, which can in turn build optimal trusses from a given initial truss. The direct representation consumes many more evaluations before reaching an optimum with similar fitness, which is explained by the higher number of dimensions of the problem. But one fitness evaluation, in the case of the direct approach, only consumes a single call to the FEA routine, whereas a single evaluation with *EiffelBlob* can consume up to 4096 calls to the FEA routine. Both approaches produce trusses of similar weight. The direct approach tends to mostly adjust the beams radius, whereas the *EiffelBlob* approach uses both the truss geometry and beams section to

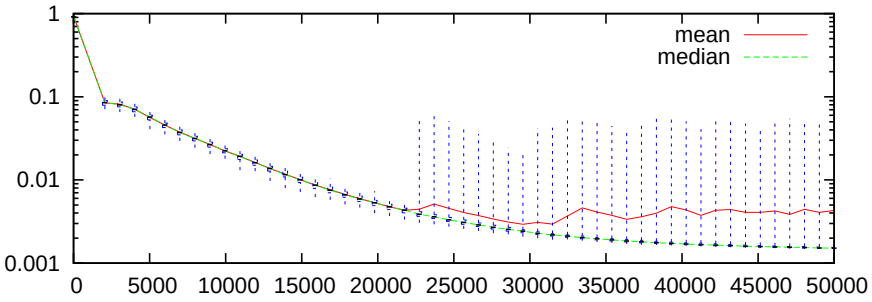


Figure 5.11: *8×6* cantilever optimization with direct representation. X axis is number of fitness evaluations, Y axis is normalised weight with log scale. The dotted bars show the standard deviation.

minimize the structure weight. Here again, some more complex problems should be used to highlight the differences between both approaches even more.

5.3.3 Properties of the Developement Process

We tested the ability of the optimized *EiffelBlob* development process to optimize trusses that are *different* from the ones used during the optimization, as initial point of the development process. Three different scenarios have been set up. In the *Perturbed Geometry* scenario, 10 different initial geometries are used, where the joint positions in the 8×6 truss are randomly drawn within a disc of radius 0.05 around their initial position (figure 5.12). In the *Load Increase* scenario, the 8×6 cantilever is now loaded with a load of $12000N$ (instead of $6000N$). And in the *Larger Dimensions* scenario, a 9×6 (regular) initial geometry is used instead of the initial 8×6 cantilever.

The best individuals for each of the 20 runs of the 8×6 cantilever problem presented in previous section are re-evaluated using these perturbed experimental conditions, but no further evolution occurs. The development is performed during at most 4096 steps. The resulting fitness is then compared to that of the **optimization** by the direct approach: for each scenario, the direct approach is run 20 times from the same geometries for 50000 iterations.

Tables 5.3.3 and Table 5.3.3 show the results for each of the problems, with the following comparison protocol: given a developped *EiffelBlob* individual, the weight obtained after development is compared to that obtained after optimization by the direct method for the same problem. If the beam stress is above the threshold or the developement does not stop, the CAE solution is considered invalid; if it is valid, if the weight obtained by the *EiffelBlob* method is lower or equal than that of the direct method (up to $2\times$ the standard deviation of the weight obtained by the 20 direct approach run), the CAE solution is considered 'optimal' (and suboptimal otherwise).

While results do depend on the actual perturbed scenario, a vast majority

	Opt.	Subopt.	Invalids
Perturbed Geometry	8	6	6
Load Increase	20	0	0
Larger Dimensions	8	8	4

Table 5.1: Capabilities of the best controllers of the 20 runs, within an altered version of the physical environment used during their optimization.

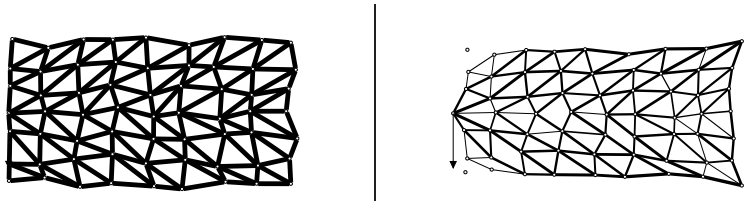
	4×4	*8×6*
EiffelBlob	5.69 – 5.06	1.58 – 1.48
Direct encoding	4.98 – 4.92	1.50 – 1.46

Table 5.2: Median and best ever of the best solutions weight $\times 10^{-3}$

of individuals did achieve valid, if not optimal, results. Moreover, individuals with perfect development are rather frequent, and advocate the relevance of the developmental approach with regards to robustness towards noisy initial conditions and truss size scalability: to some extent, less than 4096 FEAs are needed to develop a truss in a novel setup (and very often much less), while close to 50000 are needed for the direct approach whenever a new experimental setup is presented (see Section 5.3.2). Due to numerical stabilities issues of our implementation (the mass-spring system), we were not able to experiment with much larger structures, say 80×60 trusses.

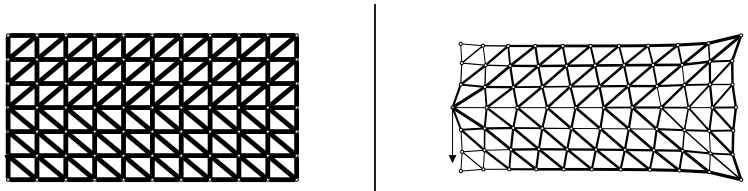
5.4 Conclusion

We presented *EiffelBlob*, a cellular development model for Structural Truss Design. This model is very similar to the one devised in the chapter 4, and can be viewed as an application of the embryogenic flags. However, apart from the different lattice structures, triangular versus square, the main difference is the coupling with a physical simulation. On a classical truss design problem, we



Initial state | Final state

Figure 5.12: On the left, a perturbed initial 8×6 cantilever is used to bootstrap an already optimized *EiffelBlob* developement process. The result of the developement is shown on the right.



Initial state | Final state

Figure 5.13: On the left, a 9×6 cantilever, instead of the 8×6 cantilever, is used to bootstrap an already optimized *EiffelBlob* developement process. The result of the developement is shown on the right.

showed that *EiffelBlob* is able to optimize a truss structure and demonstrated that the solution has some interesting properties. An optimized parameter set is able to optimize various trusses, showing in practice both the applicability of the cellular approaches for Evolutionnary Design, and the fullfilling of their promises. The reusuability capability on different context shows again the importance of the stopping criterion, where we search and optimize a fixed-point state rather than a transient state.

Nevertheless, this should be considered as work in progress, where several issues are yet to be adressed. The trusses handled by *EiffelBlob* are constrained by the underlying 2D triangular lattice. Fortunately, in the conclusion of chapter 4, we already devised an extension of our embryogenic flag model that should be able to handle any kind of cell communication graphs, possibly one that change over the time. Furthermore, the route from two to three dimensions does not seem to be that long, at least from a theoretical point of view, as our model does not use any particular properties of the 2D Euclidean geometry. Another insighful critic by mechanical experts is that the beams radius are considered here as continuous values, whereas in reality, designers can just choose among a finite set of radius (the ones that are commonly produced by the forges). From our point of view, this just a matter off expression function which outputs discrete rather than continuous values, and thus, not an alarming issue.

However, we believe that the main, real issue that we should worry about is the computational cost of our approach. *EiffelBlob* is simply too greedy in terms of calls to the FEM routine. The reusability capability might help to greatly pay off that cost, and here is a possible way to build on it. When evaluating a controller setting, the mechanical feedback might be computed just at one time, as a part of the initial state of the cells. Then the cells states would be updated, without any mechanical feedback. Once a stable state is reached by the cells, their expression function would output a truss transformation. At the beginning of the optimization, a few cycles of feedback/stabilization/transformation would be used per evaluation. Eventually, later on, more of such cycles per evaluations could be invested for further, finer optimizations. On a full scale usage, after the optimization, an optimized controller setting, those cycles would be used until the sequence of the transformed trusses reaches a fixed-point, or diverge to non-

optimal trusses. What would be optimized in such setting is an optimization policy, with much fewer call to the FEM routine. Thanks to the reusability capabality, performing few feedback/stabilization/transformation cycles per evaluation might be sufficient to find good policies. This is the subject of on-going work.

I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I've watched c-beams . . . glitter in the dark near the Tanhauser Gate. All those . . . moments will be lost . . . in time, like tears . . . in the rain. Time . . . to die.

Roy Batty in Blade Runner

6 Discussion and Conclusion

In this concluding chapter, we first summarize our contributions to Evolutionary Design. We focus on our work on cellular representations, revealing what we think are key points, and the current open issues. Then, we discuss some possible ways to tackle those issues.

6.1 Contributions

6.1.1 Representations for Evolutionary Design

In the introductory chapter and the survey chapters, (chapter 1 and chapter 2), we presented Evolutionary Design as the combination of an evolutionary algorithm with a representation of a physical object. We looked at the consequences of such a combination. Explicit representations, those which feature a direct relationship between the genotype size and the phenotype size (ie a bound on the phenotype size, computable solely from the genotype size), are usually adequate up to a moderate scale. For large phenotypes, implicit representations are likely to be more scalable. A general idea behind the implicit representations is to represent an object as the result of a building process. Generative approaches, like L-systems, have an interesting expressive power, but they might lead to a poor evolvability in practice. Cellular representations, where a building process is carried by many cells interacting together and with their environment, might address that evolvability problem. Moreover, if the cells are identical, then the genotype size is independent from the phenotype size.

We also underlined the strong importance of the environment awareness. Many design problems might be considerably easier when considering building processes that are aware of their environment. That environment would bring knowledge relevant to the problem at hand: mechanical stress for mechanical design, electromagnetic fields for antennae design.

6.1.2 Explicit Building Processes

Chapter 3 introduced *BlindBuilder*, a representation of a building process. This representation is quite general, working with the rather generic notions of assem-

bling operators (how to assemble parts of a constructions), and atomic elements of constructions (what are the elementary parts of a construction). A building process is then represented as a Directed Acyclic Graph, where the internal nodes are the assembling operators and the atomic elements are the atomic constructions elements. While optimization of moderate scale objects towards simple objectives was possible, we still faced the scalability issue from the representation point of view. Increasing the computational effort did little to nothing to both the scale of the candidate solutions as well as their optimality. We believe that we in fact had hit the evolvability issue and suffered from the lack of environment awareness. Instead of trying to "fix" *BlindBuilder*, we focused on a rather new approach for Evolutionary Design : cellular representations. As reviewed in chapter 2, cellular representations for Evolutionary Design have not been explored until very recently.

6.1.3 Embryogenic Flags

In order to study cellular representations, in chapter 4, we relied on the embryogenic flag problem, introduced by Miller et al. [85, 86]. This is a simple testbed that exempliflies most of the key aspects of the cellular representations. The cells are the tiles of a square grid, the flag, and they should turn into a color so that the flag matches a given pattern. The cells have all the same initial state, and the same controller. A cell controller updates its cell state according to the surrounding cell states. At some point, the cells states are interpreted as a color, giving the flag pattern. After an account of the many facets of the embryogenic problem, we reviewed the previous works on embryogenic flags.

We used three different architectures as cell controller for an embryogenic flag, through the same experimental setup. At first, we used a general recurrent neural network, with discrete time and continuous states. This choice was motivated by the generality of the computations carried out by such neural networks. The neural networks were optimized with the *NEAT* algorithm [113], a state of the art evolutionary algorithm for neural networks optimization. That way, we were able to successfully represent cellular development processes, here expressing some simple target patterns.

We then replaced general recurrent neural network with Echo State Networks (ESN) [54]. Those neural networks are defined statistically, being built according to a carefully chosen probability distribution. ESN retains most of the expressive power of the recurrent neural networks, while being optimizable by a purely parametric optimizer, CMA-ES [3] in our case. With this setup, we obtained the same performance than with the *NEAT* based approach. But where the *NEAT* optimizer can be quite difficult to tune due its high number of parameters, ESN with CMA-ES does not suffer from this issue.

Finally, we experimented a reaction-diffusion inspired approach on the embryogenic flag problem. In both previous architectures, the cell controllers were very general ones. Here, the controller architecture is more *had hoc*: the cells transmit information through a diffusion process. The state update of a cell is carried out by a simple perceptron in place of a complex and larger recur-

rent neural network. In a whole, this architecture mimics a discretization of a reaction-diffusion system. With this approach, using CMA-ES as optimizer, we could represent development processes that perform better or at least similarly than our two previous approaches on the embryogenic flag problem. However, cell states and cell controller are much simpler, going from up to forty states per cell, to three states per cell.

Is the architecture of the cells controller a critical issue ? As presented in the review of the chapter 4, many different architecture had already been proposed, and in the same chapter, we introduced three different ones. All those architectures were able to describe development processes with interesting properties. It seems that, as far the cell controller has a sufficient expressivity, its actual architecture does not matter a lot. To improve on the state of the art, and obtain a better (and controllable) scalability and robustness of the development processes, one might explore the other elements of a cellular representation. We focused on the development stopping criterion used when optimizing development processes, which lead us to some new and interesting insights.

6.1.4 Optimizing Cellular Development Processes

We underlined the strong importance of waiting for a steady state when optimizing development processes, with a simple experiment in chapter 4 Section 4.8. It might seem a bit misleading: by excluding transient states, we might in fact address a much harder optimization problem. But cellular representations are expected to provide scalable solutions. We interpret the scalability of a development process as its capacity to generate good solutions at different scales and in different conditions, e.g. in presence of noise. Steady states of a development process can be attractor states: the development will end-up to that same state even when started from different initial conditions. This definition matches the notion of attractor states in the dynamical systems theory. Some dynamical systems end up to a same regime despite different initial conditions and environment, and those regimes are resilient to noise. A scalable and robust development process encodes a physical object as an attractor state. The optimal object would be the attractor state of the optimal development process.

On the embryogenic flag problem, we showed in chapter 4 section 4.8 that keeping only development processes that lead to steady states, provides indeed robust attractor states. Those attractor states are resilient to noise up to a given (a priory unknown) level. Moreover, when reusing those convergent development processes at different scales, a similar pattern at different scales is maintained. On the other hand, keeping transient states when evaluating development processes completely breaks the promise of scalable solutions of the cellular representations. The optimization problem is made easier when evaluating the transient states of a development process. But in the end, the resulting optimized development process gives a same pattern only when used in the same conditions than those used during the optimization. Reusing a development process at different scales becomes impossible.

6.1.5 Stopping Criterion

While investigating the optimization of development processes, through the embryogenic flag problem, we introduced an original development stopping criterion. The intention of that stopping criterion is, when evaluating development processes, to reward only those that stop to a steady state. In order to detect a steady state, we compute an energy measure of the cells. If this energy measure remains constant over a time window, we consider that the development process has reached a steady state. In practice, this energy based stopping criterion fulfills its purpose : steady states are detected, transient states are discarded. Moreover, this stopping criterion worked for the four different controller architecture that have been introduced in this thesis (three in chapter 4 and one in chapter 5). Thus, the energy based stopping criterion appears to be both general and practical.

6.1.6 A Cellular Representation for Trusses

As a demonstration of the potential usefulness of cellular representations for Evolutionary Design, we devised a cellular representation for the optimization of structural trusses in chapter 5. A salient feature of this work is the use of environment feedback, as advocated in chapter 1. The demonstration aimed at a modest goal: minimizing the weight of a support structure. Each beam and node of the truss structure is a cell, which is aware of some problem-relevant information: the mechanical stress put on each beam. The cells locally alter the truss structure, and this in turn changes the repartition of the mechanical stress over the structure. With our cellular representation, we obtained the same fitness levels than those reached by an explicit approach, despite the simplicity of the cell controllers. Thanks to our energy based development stopping criterion, the obtained solution were somewhat scalable. A fair fraction of the optimized controllers, when reused on slightly different initial trusses, where still able to give good solutions (in comparison to an explicit approach). Thus, our work pointed out a feature of the cellular representations: the ability to obtain scalable solutions.

An issue, however, is the computational cost of an evaluation. Our cellular representation can encode any truss structure (up to some topological constraints), whatever its size, into about one hundred parameters. The explicit representation does not scale at all, needing more parameters as the truss size increases. As a consequence, the cellular representation needed less evaluations from the optimizer than the explicit representation, to reach the same fitness levels. But that does not mean that the cellular representation is a more efficient one. The evaluation cost was indeed dominated by the computation of the mechanical stress within the FEM routine. In this work on truss structures, the cellular representation needs thousands of call to the FEM routine per fitness evaluation, whereas the explicit one needs just one. Thus, the cellular representation can be more efficient in term of computational resources, but only for really large truss structures. Furthermore, although we do not have a general

answer to the guarantee of scalable solutions, we have some hints about the way to tackle the computational cost problem.

6.1.7 Summary

In the chapter 1, we left the end of the section 1.3.7 with questions about the cellular representations.

1. Which update functions are suitable? Is there a universal family of update functions? Or are they strongly application dependent?
2. When to stop the iterative update process? How to evaluate the effectiveness of a building process?
3. How complex are typical update functions? At which problem size is a cellular representation genotype more compact than an explicit representation genotype?
4. Are cellular representations truly evolvable?

We are now able to answer to those questions :

1. The experiments on the embryogenic flags and the trusses used completly different update functions, tailored for the problem. A strong linkage with the problem physic is advocated, which is clearly contradictory with the belief oin the existance of an universal family of update functions.
2. It makes sense to stop an iterative process when it reached a steady state. This steady state is the result of the process and it is what is evaluated.
3. The experiments on the embryogenic flags and the trusses used really simple update functions. Indeed, in both problems, cellular representations are more compact than direct representations even for moderate scales of the studied problems.
4. Although the relationship between the genotype and the phenotype is rather complex and indirect, cellular representations are able to provide design with a quality equal to direct approaches.

6.2 Perspectives for cellular representations

In this thesis, some advantages of cellular representations have been presented: they are scalable representations and provide scalable solutions. Moreover, they are easy to couple with physics simulations. This flexibility comes with its own drawbacks, the computational cost. The cellular representations encode objects that are obtained as the final result of the evolution of a dynamical system, made of identical distributed entities, the cells. The cells are usually supposed to be in massive quantities: cellular representations address the issue

of large phenotypes. The growth process must be simulated, until a steady state is reached. Furthermore, this growth process is likely to be coupled with a physics simulation. High numbers of development steps (waiting for a steady state) with high number of cells (because we are interested in large phenotypes): evaluating growth processes is likely to be very expensive. And since we are using evolutionary algorithms to optimize growth processes, some high numbers of evaluations are usually needed. We believe that this is the current main Achilles' heel of cellular representations: a potentially outrageous computational cost.

6.2.1 Current Issues

From this observation, we can sketch the disadvantages of cellular representations. The first consideration is the circumstances for wich they are used. In some fields, the exploratory power of a representation is much more important than its evolvability, as in Architectural Design for instance. In this context, the point is not to find optimal designs, but to explore a world of random yet sound designs. This is no longer an optimization process, because the user of the representation has no real idea of which design is optimal, and which is not. It is more about sampling a design space, to have an idea of what this design space can contain. In that sense, exploration rather than optimization, purely generative systems seem better. Grammar-based approaches can generate huge and very complex objects without affording the cost of a full fledged cellular representation. Parametric L-systems can be coupled with an environment as well (*Genr8* [43, 44, 45]), so that the design is influenced by this environment. Although cellular representations can play that role of generative representations [32], their computational cost might temper their attractiveness. Even for pattern generation, other approaches have a good expressivity while being much cheaper in term of computational cost (Evolution of function compositions, as proposed by Stanley with *NEAT* for instance [114]).

Another consequence of the potentially high computational cost of the cellular representations is that they might be interesting only for very large phenotypes. Although cellular representations feature genotype size independent from the phenotype size, a phenotype size dependent approach could be more competitive up to relatively large phenotypes. This is clearly the case in the work presented in the chapter 5. From a computational effort point of view, the cellular representation is way more expensive than the explicit representation. Each evaluation with the explicit representation needs a single physical update (to compute the phenotype fitness), while the cellular representation needs thousands of them (to compute the development process). The explicit representation needs more evaluation to find good enough solutions with larger phenotypes. But the cellular representation needs really large trusses to fadeout the cost of the thousands physics updates. Thus, for moderate scale structures, when the scalability of the solutions is useless, an explicit representation is at the moment a better and safer bet.

6.2.2 Future Directions

As stated in the previous subsection, cellular representations suffer from a potentially high computational cost of the evaluation function. Such high cost has two main sources. The first one comes from the distributed computations over massive quantities of cells, into a simulated environment. The second source of computational cost is the need to wait to for a steady state. Numerous development steps might be needed to reach such a steady state, if such a state exists. And to make things worse, the whole computational cost of the evaluation function is the product of the costs of those two parts: the number of development steps, and the numbers of cells. In the way to make cellular representations a mature approach, we think that the main obstacle is the cost of the evaluation function.

Against the high number of cells, we do not have a generic answer, out of the parallelization. All the cells share the same controller, but have different states, this fits perfectly with the SIMD (Single Instruction Multiple Data) computational model. There is currently a massive trend around the Graphical Processing Unit (*GPU).* Those devices contain hundreds of processors that can process small programs (usually a few kilobytes), working on small amounts of data (usually, also a few kilobytes) in parallel. Furthermore, the data should be arranged in a very patterned fashion. Although quite constraining, such setup perfectly suits the cellular automata context. On the experiments presented in this thesis, the number of states, the complexity of the controllers, the elementary cell neighbourhoods would fit easily into the limits of a current *GPU*. But this solution does not tackle the computational cost of a cellular representation, it could just allow us to cope with it.

We propose a more subtle (and speculative) way to offset the computational cost of a cellular representation. With the energy based stopping criterion introduced in chapter 4 section 4.4.2, an upper bound S_{max} on the number of development steps has to be provided. Using the energy based stopping criterion in an evaluation function is thus equivalent to selecting all the development processes which admit a steady state bellow S_{max} development steps. A first idea is to dynamically adapt this S_{max} threshold. One might start the optimization run with a low S_{max}. The first solutions of the optimization run are likely to be mediocre, but at least they will consume few development steps. Then, because of the low S_{max}, the quality of the solutions will stall: S_{max} should be increased to allow for more complex development processes. By adapting S_{max} to some kind of success rate of the optimization (like the likelihood to produce fitter individuals), long development processes can be optimized at the fraction of the cost of an optimization with a fixed S_{max}. Since we are adapting a parameter of the stopping criterion, we should not loose anything with respect to the robustness and the scalability of the best solutions. Thus, longer development processes might be affordable with a dynamic tuning of S_{max} versus a fixed S_{max}, and can lead to more refined solutions at a reduced cost.

An even more speculative but also potentially more effective idea is to introduce development stages, as done by Federici et al. [106]. Again, here, one

might start the optimization run with a low S_{max}. Once the optimizer is not able to produce fitter development processes any more, the fitter development process found so far is *frozen*. The *frozen* development process steady state is then used as the initial state of the further candidate development processes. By using successive *freezing* of the best development process found so far when the optimizer is stalling, we are building *stages*. The optimization is carried out stage by stage, and each stage consumes relatively few development steps. That way, very long development processes can be explored at a much reduced cost. The scalability of the development processes is likely to be preserved by this approach, since only steady states are kept to start a new stage. There are however two fundamental differences between what we propose and Federici's approach. First, the introduction of a new development stage would here be fully automated. There are no numbers of stages to setup, and no need to a priori decide when a new stage should be started. With our approach, the different stages could be of arbitrary length, introduced at arbitrary moment, entirely automatically. Second, we still guarantee that the development processes end up onto a steady-state. Indeed, each development stage will end up onto a steady-state with the proposed approach, as we will be using the energy based stopping criterion. Thus, we are likely to obtain development processes with a far better robustness and scalability compared to what is possible with Federici's approach. The staged development approach proposed here can reduce the number of fitness evaluations while retainning all the scalability properties, without introducing additional parameters to tune.

A

The Springy Comb

We devise here a dynamical system, the *Springy Comb*, which state is described by two vectors Y and V of dimension N. This system have a single global attractor state : X as a linear gradient from 0 to 1. As seen in Chapter 4, such a system could be a building block, at least from a theoritical point of view, to provide a fully developemental solution to the embryogenic flag. This dynamical system takes it roots from classical mechanics.

A.1 Description

On an Euclidian plane, let's take N infinite axis regulary spaced along the unit segment and orthogonal to it. The segment starts at $(0, 0)$ and ends at $(1, 0)$. With $1 \leq i \leq N$, the coordinates x_i of the N axis on the segment are

$$x_i = \frac{2i-1}{2N}$$

Each axis have a single particle of unit mass, free to move along the axis. Each particle is connected to it left and right neighbour particle by a linear spring of unit stiffness. The rest length of all the springs is $\frac{\sqrt{2}}{N}$. The left neighbour of the particle attached to the first axis (respectively the right neighbour attached to the last axis) is a particle which can't move, stuck at $(-\frac{1}{2N}, -\frac{\sqrt{2}}{2N})$. (respectively $(1 + \frac{1}{2N}, 1 + \frac{\sqrt{2}}{2N})$).

In addition, the particles are subjects to a friction force when they move along the axis, proportional to their speed. Such a force is called *fluid friction*, and acts has if the system was in a bath of liquid, dampening the whole dynamic. Let be λ the friction coefficient. By combining the Hook law and the Newton law, the positions $y_i(t)$ of the particles along their axis could be described by the following differential equation :

$$\begin{aligned}
y_i''(t) = &\quad f(y_i(t), y_{i-1}(t)) &&+ f(y_i(t), y_{i+1}(t)) - \lambda y_i'(t) \\
y_1''(t) = &\quad f(y_1(t), -\tfrac{\sqrt{2}}{2N}) &&+ f(y_1(t), y_2(t)) - \lambda y_1'(t) \\
y_N''(t) = &\quad f(y_N(t), y_{N-1}(t)) &&+ f(y_N(t), 1 + \frac{\sqrt{2}}{2N}) - \lambda y_N'(t)
\end{aligned}$$

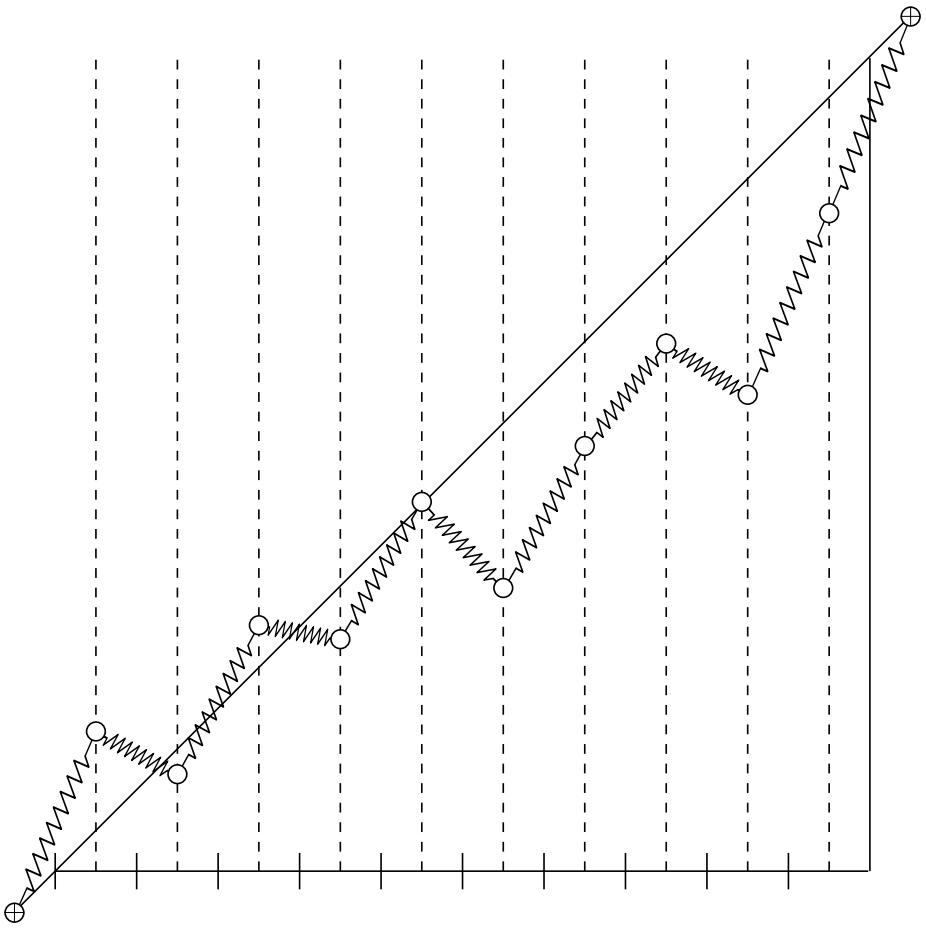


Figure A.1: The particles can move only along their dotted axis, and are linked together by springs. The leftmost and the rightmost particles are fixed, along the diagonal of the unit square. The system admits a unique null energy configuration : when the particles are on the diagonal of the unit square.

$$f(u,v) = (v-u)\left(1 - \sqrt{\frac{2}{N^2(v-u)^2+1}}\right)$$

Thus, the state of the system is fully determined by the vertical positions Y and velocities V of the particles. All the potential energy E_p of the system is stored in the springs. When all the particles are along the diagonal of the unit square, all the springs are at their rest length $\frac{\sqrt{2}}{N}$, thus the potential energy is null. In such a situation, the particles are still : this a fixed-point of the system, the unique one with a null potential energy.

A.2 Implementation

We implemented the *Springy Comb*as a 1D continuous state cellular automaton. Each cell state would be the position and the speed of a single particle : the i^{th} cell for the i^{th} particle. The update rule is simply a time integration scheme of the differential equations of the dynamical system. We used an order 4 Runge-Kutta integration scheme, with a time step of $\Delta_t = 10^{-2}$ and a friction coefficient $\lambda = 10^{-3}$. All the computations were done in *double* precision. For various numbers of cells and initial configurations, the automaton always converged close to the null potential energy fixed-point. Numerical inaccuracies introduced by the time discretization could prevents perfect convergence. Lower Δt can help to improve accuracy, as well as low (but non null) λ, at the expense of the computational time. As a consequence of the numerical inaccuracies, altough the potential energy should drop to zero, it might just remain close to it. Thus, a robust stopping criterion is to check the variance of the potential energy over a time window. If the variance is very low (10^{-10} in our experiments), then the system is likely to have reached it fixed point.

B

Blocks stacks physics

B.1 The model

We consider stacks made of identical rectangular blocks, with the same height h and length 1. Those blocks have an uniform density, then they are all of the same weight. As a further simplification, we ignore any friction phenomenon. The blocks are indestructible and non deformable. Two blocks could touch together but their interiors can't intersect. As a consequence, a stack of n blocks, resting on a perfectly flat and non deformable floor, is completely specified by the coordinates list $(x_1, y_1), \ldots, (x_n, y_n)$ of the centers of the blocks $\{B_1, \ldots, B_n\}$.

B.2 Force, equilibrium and balance

Most of the problems about blocks stack design revolve around *balanced* stacks, the stacks that don't fall apart. Because the blocks are of equal dimensions, a single block has at most two contacts above, and two contacts below. A single contact between two blocks could be completly specified by two forces vectors, as shown on Figure B.1. Let define B_i/B_j where $0 \leq i,\ j \leq n$ as the block B_i rests on B_j. Let name a_{ij} and b_{ij} the endpoints of the interval contact between the block B_i and B_j.

The sum of the forces applied on a single block, at equilibrium should be 0. We have the forces from the at most two blocks above, the forces from the at most two blocks below, and the gravity action. Because the ratio between the forces the only really important information, their actual value is not required.

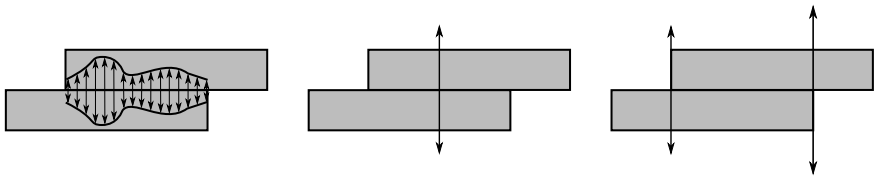


Figure B.1: Equivalents sets of forces between 2 blocks at equilibrium

That's why we will fix to 1 the force on a single block due to the gravity. Let define f_{ij}^0 and f_{ij}^1 the resultat forces that act between B_i and B_j at respectively a_{ij} and b_{ij}. Then we can write :

$$\sum_{j:B_i/B_j} (f_{ij}^0 + f_{ij}^1) - \sum_{k:B_k/B_i} (f_{ki}^0 + f_{ki}^1) = 1$$

All the resultant forces points towards the ground, so they have the same sign :

$$f_{ij}^0, f_{ij}^1 \geq 0$$

The contact forces compensate each others, so we have :

$$\sum_{j:B_i/B_j} (a_{ij} f_{ij}^0 + b_{ij} f_{ij}^1) - \sum_{k:B_k/B_i} (a_{ki} f_{ki}^0 + b_{ki} f_{ki}^1) = x_i$$

In the way we written the equilibrium equations, all the forces should remains positives. Then, for a block stack of n blocks, it equilibrium is described by $2n$ equations linear system with at most $4n$ variables and $6n$ constraints. This system is underdefined : multiple forces and moments sets can satisfy it. Finding such a set is possible with linear programming solvers.

C NEAT

NEAT is an evolutionary algorithm to optimize both the topologies and the parameters of neural networks, introduced by Kenneth Stanley. It name stands for *Network Evolution Of Augmented Topologies*, but indeed, it could be used to optimize labeled graphs, though neural networks was the original focus. *NEAT* works by applying gradual, neutral variations to neural networks. But while most of the evolutionary algorithms focus mainly on the variation operators and the representations, *NEAT* also relies a sophisticated generational operator.

C.1 The representation

A *NEAT* genotype is an array of *nodes*, and an array of *links*. A link represents an edge of the neural network : it source node and it destination node, it parameters (usually, the weight of a synapse). Additionnaly, a link has a *history marker*, and an *expression marker*. The *history marker* is a number that works as a timestamp : it denotes when the marked link was introduced during the optimization. Links added at latter iterations of *NEAT* receives history markers with greater values. The history markers are used by the generational operator, which will be detailed latter. The *expression marker* is a boolean value, specifying whever the marked link will be found in the phenotype or not. A link is said to be *enabled* when it expression marker allows it to be expressed in the phenotype, else it is *disabled*.

The phenotype is the neural network which will be evaluated by the evalution function. It is built from the nodes and links arrays, with respect to the *expression marker*. *NEAT* is an explicit representation, each node and link of the neural network is present in the genotype.

C.2 The variation operators

NEAT relies on specific variation operators. The mutations are conceived in order to alter as less as possible the functionality of a neural network : they are neutral, or almost neutral mutations. In the other end, the crossover operator

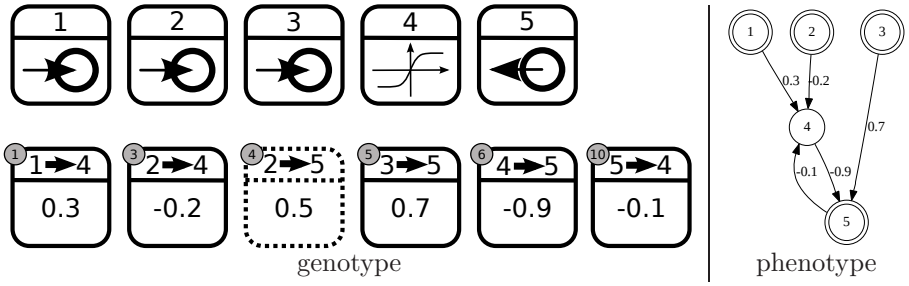


Figure C.1: A *NEAT* genotype and it corresponding phenotype. The top row represents the nodes array, while the bottom row represents the links array. The nodes 1, 2 and 3 are inputs, the node 5 is an output, and the node 4 is an hidden neuron with a sigmoid transfert function. Each link of the links array have it history marker, as a little circled number on the top-left corner. The link from node 2 to node 5 is disabled, so it is represented with a stippled box.

build new candidate solutions out of two existing candidates, and is very likely to produce neural networks with completly new functionalities.

C.2.1 Mutations

All the mutations operators use uniform random selection over nodes or links. Sometimes, one might need to maintains some topological constraints in the candidate networks. As instance, we might want only non-reccurent neural networks. One way is to apply a mutation, and if the topological constraints are violated in the mutated neural network, then the mutation is retried until the constraints are enforced. After several retries, then the mutation is aborted. That approach might have a bias towards the easiest mutations. An unbiased approach (but eventually computationally expensive) is to enumerate all the possible mutations that does not violate the topological constraints, and to randomly pick one and applying it.

- *Node addition* This operator add a node to the node array of a genotype, for instance the node with index k. Then, a link $A_{i,j}$ is randomly selected from the active links of the genotype, and is set inactive. Two active links, $B_{i,k}$ and $C_{k,j}$, are added to the genotype. The parameters of the new node and the two new links are set in order to be a functional equivalent of the disabled link $A_{i,j}$.

- *Link addition* This operator add a link to the link array of a genotype. A node pair is randomly selected from the set of the non-linked nodes pair, and the link will use that pair as source and destination. The link is enabled, and it parameters are set such as it the functionality of a null link.

- *Link enabling and disabling* The expression marker of a randomly selected link is flipped.
- *Parameters pertubation* A randomly selected node or link undergoes a pertubation of it parameters, according to a probability distraibution. Usually, a zero-centered Gaussian distribution with a fixed standard deviation is employed.

C.2.2 Crossover

C.3 The generational operator

Previous evolutionary algorithm to optimize neural networks has early convergence problems. The algorithms converged relatively fast with much ease, but towards uninteresting and bad solutions. The identified cause of that issue is a lack of diversity in the population : the candidate solutions are more or less the same after a few generations. *NEAT* tackles that problem by using a population clustering strategy. The population is clustered according to a genotype distance measure. Then, a classical generational operator is applied within each cluster. The cluster are tracked over the generations, if the mean fitness of a given cluster is stalling, then the whole cluster is sent to oblivion.

C.4 Issues

NEAT has a lot of non trivial parameters to set, which happen to be higly problem dependent : the differents type mutation probability, the parameters of the distance measure for the clustering, the distance threshold for the clustering, etc... Some of those parameters are good candidates to deterministic adaptation. We had some success by using a PID approach to regulate the clustering threshold, to maintain a given range of number of clusters. Also, the links parameters mutation strength might be regulated by a cluster-wise 1/5 rule. But there is still a lot of parameters to tune. We tune the parameters on test problems : the XOR and the inverse pendulum problem from the original *NEAT* implementation.

Bibliography

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. K. Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[2] D. Andre and J. R. Koza. A parallel implementation of genetic programming that achieves super-linear performance. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, volume III*, pages 1163–1174. CSREA, 1996.

[3] A. Auger and N. Hansen. A restart cma evolution strategy with increasing population size. In *Congress on Evolutionary Computation*, pages 1769–1776. IEEE, 9 2005.

[4] S. Babinec. Evolutionary optimization methods in echo state networks. In *6th Czech-Slovak Workshop on Cognition and Artificial Life*, 2006.

[5] T. Back, D. B. Fogel, and Z. Michalewicz, editors. *Advanced Algorithms and Operators*. IOP Publishing Ltd., Bristol, UK, UK, 1999.

[6] P. J. Bentley and J. P. Wakefield. Generic evolutionary design. In P. Chawdhry, R. Roy, and R. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 289–298. Springer, june 1997.

[7] H.-G. Beyer and U.-M. O'Reilly, editors. *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*. ACM, 2005.

[8] J. Bongard. Evolving modular genetic regulatory networks. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1872–1877, Washington, DC, USA, 2002. IEEE Computer Society.

[9] J. Bongard and C. Paul. Investigating morphological symmetry and locomotive efficiency using virtual embodied evolution. In J.-A. M. et al., editor, *From Animals to Animats: The Sixth International Conference on the Simulation of Adaptive Behaviour*, 2000.

[10] J. C. Bongard and R. Pfeifer. Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 829–836, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[11] M. Cattolico, editor. *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006*. ACM, 2006.

[12] A. Chavoya and Y. Duthen. Using a genetic algorithm to evolve cellular automata for 2d/3d computational development. In Cattolico [11], pages 231–232.

[13] A. Chavoya and Y. Duthen. An artificial development model for cell pattern generation. In *Australian Conference on Artificial Life (ACAL 2007)*. Springer, december 2007.

[14] A. Chavoya and Y. Duthen. Use of a genetic algorithm to evolve an extended artificial regulatory network for cell pattern generation. In Lipson [73], pages 1062–1062.

[15] L. Clement and R. Nagpal. Self-assembly and self-repairing topologies, 1 2003.

[16] C. A. C. Coello, D. A. V. Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.

[17] R. Connelly and W. Back. Mathematics and tensegrity. *American Scientist*, 86(2):142–151, 1998.

[18] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[19] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

[20] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, 2001.

[21] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In M. S. et al., editor, *Proceedings of the* 6^{th} *Conference on Parallel Problems Solving from Nature*, pages 849–858. Springer-Verlag, LNCS 1917, 2000.

[22] K. Deb and S. Gulati. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elem. Anal. Des.*, 37(5):447–465, 2001.

[23] K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors. *Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004, Proceedings, Part I*, volume 3102 of *Lecture Notes in Computer Science*. Springer, 2004.

[24] D. Dickmanns, J. Schmidhuber, and A. Winklhofer. Der genetische Algorithmus: Eine Implementierung in Prolog. Fortgeschrittenenpraktikum, Institut für Informatik, Lehrstuhl Prof. Radig, Technische Universität München, 1987.

[25] M. Ebner. Evolutionary design of objects using scene graphs. In C. Ryan, T. Soule, M. Keijzer, E. P. K. Tsang, R. Poli, and E. Costa, editors, *EuroGP Proceedings*, volume 2610 of *Lecture Notes in Computer Science*, pages 47–58. Springer, April 2003.

[26] P. Eggenberger-Hotz. Evolving morphologies of simulated 3d organisms based on differential gene expression. In P. Husbands and I. Harvey, editors, *Proceedings of the 4th European Conference on Artificial Life (ECAL97)*. MIT Press, Cambridge, MA, 1997.

[27] P. Eggenberger-Hotz. Comparing direct and developmental encoding schemes in artificial evolution a case study in evolving lens shapes. 2004.

[28] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[29] N. S. Estévez. Functional Blueprints: A Dynamical Systems Approach to Structure Representation. Master's thesis, Mechanical Engineering Dept., Cornell University, 2007.

[30] N. S. Estévez and H. Lipson. Dynamical blueprints: exploiting levels of system-environment interaction. In Lipson [73], pages 238–244.

[31] N. Fatès and M. Morvan. An experimental study of robustness to asynchronism for elementary cellular automata. *Complex Systems*, 16:1–27, 2005.

[32] K. W. Fleischer. *A multiple-mechanism developmental model for defining self-organizing geometric structures*. PhD thesis, California Institute of Technology, Pasadena, CA, USA, 1995. Adviser-Alan H. Barr.

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

[34] P. Funes and J. B. Pollack. Computer evolution of buildable objects. In P. Husbands and I. Harvey, editors, *Fourth European Conf. on Artificial Life*, pages 358–367, Cambridge, MA, 1997. MIT Press.

[35] C. Gagné and M. Parizeau. Open BEAGLE: A new C++ evolutionary computation framework. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 888, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[36] D. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, (3):493–530, 1989.

[37] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, January 1989.

[38] T. G. W. Gordon and P. J. Bentley. Bias and scalability in evolutionary development. In Beyer and O'Reilly [7], pages 83–90.

[39] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Ecole Normale Supï¿½rieure de Lyon, France, 1994.

[40] J. F. Hall. Fun with stacking blocks. *American journal of physics*, 73(12):1107?1116, 2005.

[41] H. Hamda, F. Jouve, E. Lutton, M. Schoenauer, and M. Sebag. Compact unstructured representations in evolutionary topological optimum design. *Applied Intelligence*, 16:139–155, 2002.

[42] H. Hamda and M. Schoenauer. Adaptive techniques for evolutionary topological optimum design. In *In I. Parmee, editor, Evolutionary Design and Manufacture*, pages 123–136. Springer, 2000.

[43] M. Hemberg. GENR8 - A design tool for surface generation. Master's thesis, Department of Physical Resource Theory, Chalmers University, Sweden, June 29 2001.

[44] M. Hemberg and U.-M. O'Reilly. GENR8 - using grammatical evolution in A surface design tool. In A. M. Barry, editor, *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 120–123, New York, 8 July 2002. AAAI.

[45] M. Hemberg and U.-M. O'Reilly. Using generative growth systems to design architectural form. In M. Bedau, P. Husbands, T. Hutton, S. Kumar, and H. Sizuki, editors, *Workshop and Tutorial Proceedings Ninth International Conference on the Simulation and Synthesis of Living Systems(Alife XI)*, pages 33–36, Boston, Massachusetts, 12 Sept. 2004.

[46] G. Hinton and S. Nowlan. How learning can guide evolution. *Complex Systems*, 1(495-502), 1987.

[47] T. Hohm, M. Egli, S. Gaehwiler, S. Bleuler, J. Feller, D. Frick, R. Huber, M. Karlsson, R. Lingenhag, T. Ruetimann, T. Sasse, T. Steiner, J. Stocker, and E. Zitzler. An Evolutionary Algorithm for the Block Stacking Problem. In N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2008.

[48] G. S. Hornby. *Generative Representations for Evolutionary Design Automation.* PhD thesis, Brandeis University, Dept. of Computer Science, Boston, MA, USA, Feb. 2003.

[49] G. S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In Beyer and O'Reilly [7], pages 1729–1736.

[50] G. S. Hornby, H. Lipson, and J. B. Pollack. Evolution of generative design systems for modular physical robots, 5 2001.

[51] G. S. Hornby and J. B. Pollack. Evolving L-systems to generate virtual creatures. *Computers and Graphics*, 25(6):1041–1048, 2001.

[52] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, pages 359–366, 1989.

[53] H. Iba. Random tree generation for genetic programming. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 144–153, London, UK, 1996. Springer-Verlag.

[54] H. Jaeger. The Echo State approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001.

[55] H. Jaeger. Short term memory in echo state network. Technical Report GMD Report 152, German National Research Center for Information Technology, 2001.

[56] H. Jaeger. Tutorial on training recurrent neural networks. Technical report, GMD Report 159, Fraunhofer Institute AIS, 2002.

[57] H. Jaeger, H. Haas, and J. C. Principe, editors. *NIPS 2006 Workshop on Echo State Networks and Liquid State Machines*, 2006.

[58] T. Jakobsen. Advanced character physics. In *GDC 2001*, 2001.

[59] F. Jiang, H. Berry, and M. Schoenauer. Unsupervised learning of echo state networks: balancing the double pole. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 869–870, New York, NY, USA, 2008. ACM.

[60] L. Kallel and M. Schoenauer. Alternative random initialization in genetic algorithms. In *Proceedings of the 7 th International Conference on Genetic Algorithms*, pages 268–275. Morgan Kaufmann, 1997.

[61] C. Kane and M. Schoenauer. Topological optimum design using genetic algorithms. *Control Cybernetic*, 25:1059–1088, 1996.

[62] T. D. S. Kathleen T. Alligood and J. A. Yorke. *Chaos. An introduction to dynamical systems.* Springer-Verlag, 2000.

[63] R. Kicinger, T. Arciszewski, and K. A. De Jong. Morphogenesis and structural design: cellular automata representations of steel structures in tall buildings. In *Proceedings of the Congress on Evolutionary Computation (CEC'2004)*, pages 411–418, Piscataway, NJ, 6 2004. IEEE Press.

[64] R. Kicinger, T. Arciszewski, and K. A. De Jong. Evolutionary computation and structural design: A survey of the state of the art. *Computers & Structures*, 83(23-24):1943–1978, 2005.

[65] R. Kicinger, T. Arciszewski, and K. D. Jong. Evolutionary computation and structural design: A survey of the state-of-the-art. *Computers & Structures*, 83(23-24):1943–1978, 2005.

[66] R. Kicinger, T. Arciszewski, and K. D. Jong. Parameterized versus generative representations in structural design: an empirical comparison. In Beyer and O'Reilly [7], pages 2007–2014.

[67] T. Kowaliw, P. Grogono, and N. Kharma. Environment as a spatial constraint on the growth of structural form. In Lipson [73], pages 1037–1044.

[68] T. Kowaliw, P. Grogono, and N. N. Kharma. Bluenome: A novel developmental model of artificial morphogenesis. In Deb et al. [23], pages 93–104.

[69] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[70] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

[71] P. Krcah. Evolutionary development of robotic organisms. Master's thesis, Faculty of Mathematics and Physics, Department of Software and Computer Science Education, Charles University, Czech Republic, 2007.

[72] D. S. Linden. *Automated design and optimization of wire antennas using genetic algorithms*. PhD thesis, Massachusetts Institute of Technology, 1997. Supervisor-Frederic R. Morgenthaler and Supervisor-Mark J. Jakiela.

[73] H. Lipson, editor. *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*. ACM, 2007.

[74] J. Lohn, J. Crawford, A. Globus, G. S. Hornby, W. Kraus, G. Larchev, A. Pryor, and D. Srivastava. Evolvable systems for space applications. In *International Conference on Space Mission Challenges for Information Technology*, 2003.

[75] J. Lohn, G. S. Hornby, and D. Linden. An evolved antenna for deployment on nasa's space technology 5 mission, 2004.

[76] J. D. Lohn, W. F. Kraus, D. S. Linden, and S. Colombano. Evolutionary optimization of yagi-uda antennas. In *ICES '01: Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, pages 236–243, London, UK, 2001. Springer-Verlag.

[77] J. D. Lohn, D. S. Linden, G. S. Hornby, W. F. Kraus, and A. Rodriguez-Arroyo. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In *EH '03: Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, page 155, Washington, DC, USA, 2003. IEEE Computer Society.

[78] S. Luke and L. Panait. Lexicographic parsimony pressure. In *GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 829–836. Morgan Kaufmann, 2002.

[79] S. Manos, M. C. J. Large, and L. Poladian. Evolutionary design of single-mode microstructured polymer optical fibres using an artificial embryogeny representation. In Lipson [73], pages 2549–2556.

[80] S. Manos, L. Poladian, P. J. Bentley, and M. Large. Photonic device design using multiobjective evolutionary algorithms. In C. A. C. Coello, A. H. Aguirre, and E. Zitzler, editors, *Proceedings of Evolutionary Multi-Criterion Optimization, Third International Conference*, pages 636–650, march 2005.

[81] S. Manos, L. Poladian, P. J. Bentley, and M. Large. A genetic algorithm with a variable-length genotype and embryogeny for microstructured optical fibre design. In Cattolico [11], pages 1721–1728.

[82] F. Mansanne, F. Carrre, A. . Ehinger, and M. Schoenauer. Evolutionary algorithms as fitness function debuggers. In Z. W. Ras and A. Skowron, editors, *Foundation of Intelligent Systems 12th International Symposium, ISMIS99*, volume 1609 of *LNCS*. Springer Verlag, 1999.

[83] S. Meyer-nieberg and H. georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithm*. Springer, 2006.

[84] T. Miconi and A. Channon. An improved system for artificial creatures evolution. In *in Procs 10th Intl Conf on Simulation and Synthesis of Living Systems (ALIFE*, pages 255–261. MIT Press, 2006.

[85] J. F. Miller. Evolving developmental programs for adaptation, morphogenesis and self-repair. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *ECAL*, volume 2801 of *Lecture Notes in Computer Science*. Springer, 2003.

[86] J. F. Miller. Evolving a self-repairing, self-regulating, french flag organism. In Deb et al. [23], pages 129–139.

[87] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Proceedings of the European Conference on Genetic Programming*, pages 121–132, London, UK, 2000. Springer-Verlag.

[88] R. Motro. *Tensegrity: structural systems for the future*. Butterworth-Heinemann, 2003.

[89] J. D. Murray. *Mathematical Biology*, volume 19 of *Biomathematics*. Springer, New York, 1989.

[90] R. Nagpal. Self-organizing shape and pattern: From cells to robots. *IEEE Intelligent Systems*, 21(2):50–53, 2006.

[91] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[92] M. C. Ozturk, D. Xu, and J. C. Principe. Analysis and design of echo state networks. *Neural Computation*, 19(1):111–138, 2007.

[93] M. Paterson, Y. Peres, M. Thorup, P. Winkler, and U. Zwick. Maximum overhang. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 756–765, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[94] M. Paterson and U. Zwick. Overhang. In *SODA'06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 231–240. ACM Press, 2006.

[95] C. Paul, H. Lipson, and F. J. V. Cuevas. Evolutionary form-finding of tensegrity structures. In Beyer and O'Reilly [7], pages 3–10.

[96] P. Petrovic. Solving lego brick layout problem using evolutionary algorithms. 2001.

[97] M. Peysakhov. Representation and evolution of lego-based assemblies. In C. Ryan, U.-M. O'Reilly, and W. B. Langdon, editors, *Graduate Student Workshop*, pages 297–300, Las Vegas, Nevada, USA, 8 2000.

[98] M. Peysakhov and W. C. Regli. Using assembly representations to enable evolutionary design of lego structures. *Artif. Intell. Eng. Des. Anal. Manuf.*, 17(2):155–168, 2003.

[99] J. Plavcan and P. Petrovic. Direct and indirect representations for evolutionary design of objects. In Lipson [73], pages 160–171.

[100] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[101] J. B. Pollack and H. Lipson. The golem project: Evolving hardware bodies and brains. In *EH '00: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware*, page 37, Washington, DC, USA, 2000. IEEE Computer Society.

[102] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[103] A. Pugh. *Introduction to tensegrity*. Berkeley University of California Press, 1976.

[104] I. Rechenberg. *Evolutionsstrategie 94*. Frommann-Holzboog, Stuttgart, 1994.

[105] J. Rieffel. *Evolutionary Fabrication: The Co-Evolution of Form and Formation*. PhD thesis, Brandeis University, Waltham, MA, USA, 2006. Adviser-Jordan B. Pollack.

[106] D. Roggen and D. Federici. Multi-cellular development: is there scalability and robustness to gain? In X. Yao, E. Burke, J. Lozano, and al., editors, *Proceedings of Parallel Problem Solving from Nature 8, PPSN 2004*, pages 391–400, 2004.

[107] F. Rothlauf. On the locality of representations, Jan. 28 2003.

[108] F. Rothlauf. Representations for evolutionary algorithms. In M. Ebner, M. Cattolico, J. van Hemert, S. Gustafson, L. D. Merkle, F. W. Moore, C. B. Congdon, C. D. Clack, F. W. Moore, W. Rand, S. G. Ficici, R. Riolo, J. Bacardit, E. Bernado-Mansilla, M. V. Butz, S. L. Smith, S. Cagnoni, M. Hauschild, M. Pelikan, and K. Sastry, editors, *GECCO-2008 tutorials*, pages 2613–2638, Atlanta, GA, USA, 12-16 July 2008. ACM.

[109] A. R. Sanderson, R. M. Kirby, C. R. Johnson, and L. Yang. Advanced reaction-diffusion models for texture synthesis. *Journal of Graphics Tools*, 11(3):47–71, 2006.

[110] H. A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6):467–482, December 1962.

[111] K. Sims. Interactive evolution of dynamical systems. In *Proceedings of the First European Conference on Artificial Life*, pages 171–178. MIT Press, 1991.

[112] K. Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 1994. ACM.

[113] K. O. Stanley. *Efficient evolution of neural networks through complexification*. PhD thesis, 2004. Supervisor-Risto P. Miikkulainen.

[114] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8(2):131–162, 2007.

[115] T. Toffoli and N. H. Margolus. Invertible cellular automata: a review. pages 229–253, 1990.

[116] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, B 237(641):37–72, Aug. 1952.

[117] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 289–298, New York, NY, USA, 1991. ACM.

[118] P. Turney, D. Whitley, and R. W. Anderson, editors. *Evolutionary Computation Journal – Special Issue on the Baldwin Effect*, volume 4. MIT Press, 1996.

[119] J. Werfel. *Anthills built to order: automating construction with artificial swarms*. PhD thesis, Cambridge, MA, USA, 2006. Adviser-Radhika Nagpal and Adviser-H. Sebastian Seung.

[120] S. Wolfram. *A new kind of science*. Wolfram Media, 2002.

[121] L. Wolpert. Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology*, (25):1–47, 1969.

[122] A. I. Zayed. *Handbook of Function and Generalized Function Transformations*. CRC Press, 1996.

[123] Q. Zheng, O. Querin, and D. Barton. Geometry and sizing optimisation of discrete structure using the genetic programming method. *Structural and Multidisciplinary Optimization*, 31:1–2(2), june 2006.