9

Artificial Development

Simon Harding, Wolfgang Banzhaf

Computer Science, Memorial University of Newfoundland, St. John's, NL, A1B 3X5, Canada.
simonh@cs.mun.ca, banzhaf@cs.mun.ca

Summary. There is growing interest in the use of analogies of biological development for problem solving in computer science. Nature is extremely intricate when compared to human designs, and incorporates features such as the ability to scale, adapt and self-repair that could be usefully incorporated into human-designed artifacts. In this chapter, we discuss how the metaphor of biological development can be used in artificial systems and highlight some of the challenges of this emerging field.

Key words: Developmental systems, genetic algorithms, pattern formation

9.1 Introduction

Organic processes can serve as inspiration for computational systems. The design and functionality of organic systems is both a challenge and an existence proof of achievements of Nature posed to the human designer. In this context it has turned out that the level of complexity commanded in the design of natural systems is still unparalleled. To put the scalability of biological systems into perspective: Microsoft's latest version of Windows has 10^6 lines of code, some Linux distributions have 10^7 lines of computer code, but there are 10^{14} cells in the human body each much more complex than a line of code. Such complexity would be out of reach of human designers, who even struggle to keep the simplest designs and manufacturing processes defect-free. With the increasing complexity of both hardware and software, there is growing need for new design techniques that allow us to work with such complexity.

Evolutionary algorithms that mimic Darwinian evolution have provided part of the answer to this problem. Algorithms such as genetic algorithms and genetic programming [2, 7, 22] allow us to quickly search through vast search spaces, finding novel solutions to our problems. However, there are limits to how evolvable certain applications are, especially those that – with a naïve implementation – would require long genotypes. Under such circumstances

R.P. Würtz (ed.), *Organic Computing*. Understanding Complex Systems,
doi: 10.1007/978-3-540-77657-4_9,

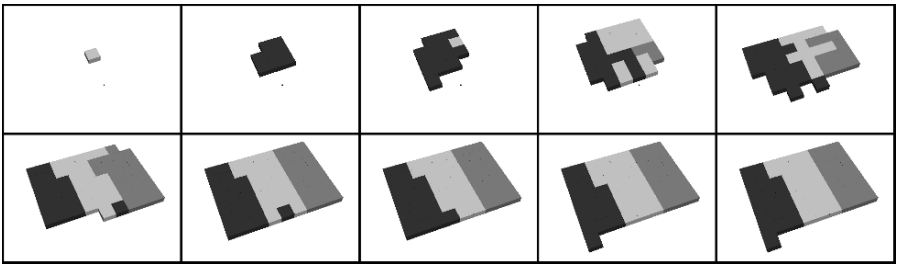

Fig. 9.1. Miller's French Flag Problem

evolution can find it difficult to fine-tune the genotype — mutations are more likely to be disruptive than beneficial and the combinatorics of search spaces is prohibitive. It would be impossible to evolve a functioning computer program with millions of lines of code, or a circuit with a hundred million components with our present-day methods. So how does nature correctly assemble a system containing trillions of cells?

Nature uses the process of development, by which a fertilized egg grows into a fully mature individual. Here we shall discuss how one can grow complicated programs and other structures by starting from an artificial zygote. For example, figure 9.1 shows a single artificial cell developing into a predefined pattern [21]. Each cell contains a computer program, discovered by evolution, that when executed contains instructions for growing the virtual organism. In this chapter, we shall provide a brief discussion of the biology of development, and shall link it to artificial development. We will also present some ideas for improving the current state of developmental models, in particular the use of physical analogies.

9.2 Embryogenesis

Embryogenesis is the first stage of the developmental process by which the embryo is formed. The development starts with the zygote, which is the fertilized ovum. The zygote undergoes a process called cleavage, where mitosis splits the cell into two (and later more) identical cells. Even at this stage, spatial patterns are being produced. Egg cells are asymmetric, and the first cleavage occurs along one axis, the second cleavage on an axis perpendicular to it. The developing embryo receives a substantial amount of information in the starting configuration from its zygote — this is called molecular prepatterning [23]. In insect ovaries, in addition to the egg, there are other cells, such as nurse cells, which produce proteins required for growth, and pass these into the egg cell. They contain maternal DNA that has an important role in the embryo's growth.

Mitosis repeats until 128 cells have been produced, at which point the next phase of development, called blastulation, occurs. At this stage, the cells are

still stem cells — they have yet to differentiate into a specialized cell type. In mammals, the cells organize into a spherical structure called a blastocoele, which subsequently develops into a blastocyst. The outer cells (called the trophoblast) will go on to form the placenta, and the inner cells (called the embryoblast) will form the embryo. Subsequently, the blastocoele forms three layers, the ectoderm, endoderm, and mesoderm, that develop into the internal organs of the animal.

The cells differentiate into their organ types, and groups form certain physical shapes. Shapes themselves are the result of a small number of different processes, governed by differences in cell adhesion and rates of differentiation. The cells can form tubes or sheets, or condense into clusters. We further discuss the role of physics in section 9.5.

Positional information, used by the genes of the embryo comes from chemical gradients that are set up by proteins from the maternal RNA. These divide the space inside the egg, and form the axes along which different developmental processes occur. In Drosophila, for example, nurse cells anchored at one end of the egg contain a gene that produces a protein (Bicoid) that diffuses across the egg, setting up a gradient. As a morphogen, the concentration of this protein determines whether certain other genes are expressed and hence control the fate of cells: A high concentration results in the formation of head and thorax. The embryo develops, however, without these features if the gradient is artificially suppressed.

The genes in the embryo are responsible for the majority of the developmental processes. They produce proteins which not only build the cellular structures, but also form part of the control process. The program stored in genes is the aspect typically replicated by evolved programs in artificial developmental processes. The genes form networks of interactions, where the behavior of one gene represses or triggers the behavior of others. However it is important to note that in addition there are maternal genes that sit at the top of this complex hierarchy of gene interactions. Maternal genes influence the zygotic genes, which in this instance are labeled gap genes, since in Drosophila a mutation in one of these genes results in a gap in the developing body plan. These genes in turn influence other layers in the gene hierarchy — at the bottom of this diagram are genes that influence the development of the wings. Effectively, each layer in the hierarchy occurs at different stages in the developmental cycle, with the lowest level occurring last, and interacting the least with the maternal RNA.

Development is sensitive to its environmental conditions. Amounts of food, water, sun and other resources play an important role in defining the growth of an animal as well as do external signals. If any of these resources is deficient, the body develops in a different way. For example, fish release hormones into the water that prevent further growth if their concentration gets too high. This prevents overuse of food resources in the wild, but also explains why the size of fish held in captivity is dependent on their tank size. Plants grow to

fit their containers — or their roots flow around rocks and other obstacles underground.

In recent years, some researchers have started to look at embryogenesis as an inspiration for the development of electronic circuits. This field of inquiry has been named *Embryonics* [24].

9.3 Scalability, Plasticity and Robustness

Multicellular organisms start from a single cell, which develops into a complete organism – potentially containing trillions of cells. Each cell contains the genetic information, encoded in DNA, that controls its properties. However, DNA does not directly specify these properties. Instead, it instructs the construction and development of a cell, using as much environmental information and material as possible. The human genome consists of about $3.2 \cdot 10^9$ base pairs, coding for approximately 25,000 genes which in turn produce 10^{14} cells. A large portion of the genome contains regulatory information.

9.3.1 Scalability

A good example of scalability in artificial development comes from neural networks. Scalability in this context means that it is possible to grow more neurons if they are needed in the neural net to solve harder problems. Specifying the properties of each cell would be computationally infeasible due to combinatorics and training algorithms like back-propagation would be unsuitable. As a result, direct evolution of weights and topology of the net sooner or later will encounter a size barrier, where training is no longer computationally practicable.

Can we write a computer program that will grow an arbitrarily sized neural net with the right properties? Is this easier than evolving the network directly? Here we are combining evolution, development and learning. Could we make the neural network perform difficult tasks? For example, say we want to write a voice recognition program. The scalable approach would require that we make it work satisfactorily on a cell phone processor, good on a desktop and amazingly well on a supercomputer. In this example, growing would allow the program to fit the processor and memory availability. Would we find that evolution/development will discover unusual topologies that one would not have considered otherwise?

Kitano was one of the first to use artificial development to produce neural networks [17]. Kitano evolved L-system rules that produced the connection matrix of neurons, specifying the network's topology and their weights. He found that, unlike conventional direct encoding, the developmental encoding scaled well. Kitano developed a method for evolving the architecture of an artificial neural network using a matrix re-writing system that manipulated adjacency matrices [17]. According to his results, this method produced results

superior to direct methods (i.e. a fixed architecture, directly encoded and evolved). It was later claimed, however, that the two approaches were of equal quality [28].

Gruau devised a graph re-writing method called cellular encoding [12]. Cellular encoding is a language for local graph transformations that controls the division of cells growing into artificial neural networks. The cells, which we can identify as nodes in the ANN, store connection strengths (weights) and a threshold value. The cells also store a grammar tree that defines the graph re-writing rules and a register that defines the start position in the grammar tree. The grammar tree was evolved using an evolutionary algorithm and the method was shown to be effective at optimizing both the architecture and weights at the same time. It scaled better than direct encoding, where all the weights had to be evolved independently [13].

Federici has successfully evolved spiking neural networks that are constructed with a developmental system [9]. Here the developmental system outperformed direct encoding by a considerable margin. However, as the parameters of neither experiment were optimized, it may not be a fair test of the algorithms' ability.

Human designs are often limited by their ability to scale and adapt to changing needs. Our rigid design processes often constrain the design to solving the immediate problem, with only limited scope for change. Organisms, on the other hand, appear to be able to maintain functionality through all stages of development, despite a vast change in the number of cells from the embryo to a mature individual. It would be advantageous to empower human designs with this on-line adaptability through scaling, whereby a system can change complexity depending on conditions. We should expect organic computing to solve a related problem: Adaptation of the complexity of algorithmic approaches to problems of variable difficulty.

9.3.2 Plasticity

In nature we find that designs scale, as is evident from the growth of animals after gestation: The various functions of the organism still work all the way from infancy to adulthood. This gives organisms the plasticity to grow to sizes that fit the environment, without sacrificing reproductive capability. The hormones in the water already mentioned limit the growth of fish, and this prevents fish from becoming too large if there is not enough room, and hence allow the fish to survive with less competition for resources. Similar effects can be found in artificial developmental systems. For example, in L-systems the models of plants can be run to any size, and still retain the same morphology or shape. L-systems were originally used to model the development of plants, however they have also been used for producing neural networks [5], protein structure prediction [8] and object design [16].

These results, both in natural and artificial contexts, give us some confidence that if we wish to produce designs with a large, and potentially chang-

ing, number of components, we should be able to utilize the developmental approach. For example, if we wish to build a control system for a large plant, such as a power station, the design should yield a stable, fail-safe control system. As with all mechanical and electrical systems, faults develop over time. It would be beneficial for the system to cope with component failure, and the system should be able to maintain its function in the event that a sensor fails. As the needs of the plant change, the system may be required to grow to accommodate new features, perhaps a boiler would be added with new sensors, actuators and terminals; could we use the principles of developmental systems to allow the control system to automatically "grow" and adapt to accommodate this? We may also find we have to deal with things that are very biological – lag in signals, different rates of signal firing, different sensor types. We therefore have to accommodate two types of change, one in the physical structure and another in the amount and quality of information in the system. By taking inspiration from the plasticity of biological systems, we may be able to produce systems that can handle such changes effectively – a task that is difficult for traditional approaches.

9.3.3 Robustness

Biological systems are remarkably tolerant to failure in individual components, and this is clearly a desirable attribute for engineered systems. The ability to regenerate lost or damaged limbs, tissues or organs is common in animals — although the abilities vary. Some animals, such as newts have the ability to re-grow entire limbs. Humans cannot regenerate limbs, however they can re-grow ribs and fingertips. The liver is also able to regenerate, and the skin is constantly being replaced.

The processes involved in development and regeneration are related. For example, during the early development of a fetus it is possible for it to fully recover from a deep cut. However, later during development the regrowth is not as effective and the fetus becomes scarred. The self-repair of the newt involves a layer of cells growing over the injured stump, which revert to stem cells. These stem cells, like those in the developing embryo, can become any cell type and allow the missing limb to re-develop.

Gerhart and Kirschner [11] describe four properties of cells that lead to developmental flexibility: (i) The destruction of a small number of cells can be tolerated, as there is enough redundancy that others in the group can replace it. (ii) As all the cells in a group perform the same action, their arrangement does not matter. (iii) Moving cells from one group to another equivalent group is possible, as the cells can adapt to the local stimuli. (iv) Finally, if an organizer, such as the bicoid gene described in section 9.2, is moved then the cells respond to their new distance from the organizer.

Which of these features are relevant and appropriate to implement in artificial development? In conventional engineering, redundancy is the normal

approach to implement fault tolerance and robustness. Although natural systems do have redundancy, for example the duplication of entire organs such as eyes and lungs, they also have the ability re-grow anything from missing cells to entire limbs. Current hardware technology does not allow a similar feat.

However, perhaps we can look forward to the time when nanobots will give hardware this ability. The application of developmental systems in pattern formation for such systems is obvious — nanobots will have limited computational and sensing abilities — much like real cells. Hence, this type of approach could be applicable in these scenarios.

Bentley shows that inherently fault tolerant computer programs can be evolved, i.e. one can damage bits of the code, and its behavior will degrade gracefully [3]. As the author notes, it is difficult to test if development gives an advantage in this case. The developmental program is longer, and hence may be more susceptible to faults (for example from faulty memory) — however, when the developmental program is corrupted, behavior degrades gracefully. The developmental program also required more computation to execute, in terms of the simulation of growing the artificial organism. Is this trade-off ultimately worth it? Bentley believes that the fragility of the solutions may be caused by the "conventional (brittle) nature of the programming language, compiler and hardware", and hence we may have to think differently about the methods of implementing these computational systems. For example, if we were to apply the cellular computing metaphor to hardware then so far all attempts have required vast amounts of hardware relative to the size of the problem being solved – and far in excess of what would be required from traditional n-module redundancy. Perhaps one could get evolution to find solutions that are more than the sum of their parts, which would gives us back the advantage? Such a system may allow us to evolve circuits with high component counts, that also have intrinsic fault tolerance – a task not yet achieved with a purely evolutionary approach.

9.4 Evolvability and search spaces

Artificial developmental systems are an example of indirect genotype-to-phenotype mapping. In development, genotypes are typically shorter than the phenotypes they represent, which means that development can be viewed as a decompression algorithm. This changes the way in which evolution tackles the search space.

For some encodings, the genotype may only be able to map to a limited part of the search space, as it may be the case that the number of states potentially represented by a short genotype is lower than the number of states in the phenotype space. For such systems, parts of the phenotype space are ignored, which can potentially benefit evolution, although care needs to be taken to ensure that potential solutions can be accessed.

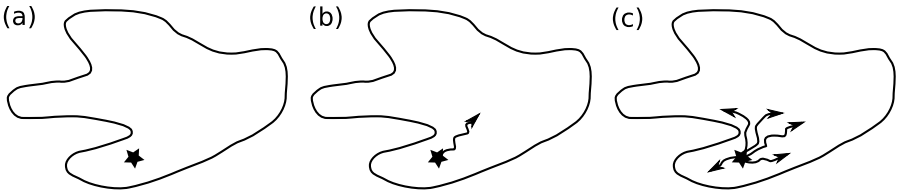

Fig. 9.2. Direct encoding, (a), only tests the search space at a single point specified by the genotype (shown as a star within the search space). A developmental encoding (b) can travel through the search space when developing. A stochastic developmental encoding (c) could take this further, and search a large area of the space, but still be the result of a single genotype.

A stochastic genotype-phenotype mapping could be viewed as a lossy decompression algorithm. The benefit here is that it enables sampling from a greater region of the phenotype space than a deterministic mapping. Natural systems are often subject to high levels of noise coming from a variety of sources, from thermal noise at a small scale to unpredictable external factors. Natural evolution has found mechanisms to cope with such noise, however in artificial systems we tend to avoid this form of stochastic behavior. Artificial evolution [30, 20, 14] shows that algorithms are not only capable of operating in such situations, but actually benefit from the presence of noise. Artificial chemistries [1] and esoteric programming languages such as "Whenever", execute program instructions in random orders - yet are still able to implement a desired computation.

One potentially undesirable feature of stochastic mapping is that for fitness evaluation each individual will need to be tested a number of times to ensure that the fitness is an accurate sample of the phenotype space that the genotype maps to. Another consequence is that it can no longer be guaranteed that a particular genotype will produce a particular phenotype, which suggests that such approaches would be most useful where a phenotype still adequately performs even if it is imperfect. On the other hand, the degree of accuracy can be graded by the amount of computational power invested into the mapping.

In a typical setting artificial development mimics the cellular structure used by nature, where cellular modules cooperate to perform a particular task. This approach has implications for the types of problems that we can attempt to solve, as it may be that some problems are not easily mapped onto this format. Indeed, human designers have difficulties in implementing such systems, and this is particularly evident in programming parallel systems, or indeed in defining rules for cellular automata. Miller reports that evolving the developmental French Flags is hard - with very few runs being successful [21]. It is still unclear whether developmental systems are easier to evolve than non-developmental systems. Although the genotype may be shorter - and hence fewer variables have to be manipulated, development can be expected to distort the fitness landscape. For example, in [15], the evolvability of a simple

developmental encoding was investigated, and it was found that evolution was less effective at finding solutions using development than with direct encoding.

Roggen and Federici compared evolving direct and developmental mappings for the task of producing specific two dimensional patterns of various sizes (the Norwegian Flag and a pattern produced by Wolfram's 1D CA rule 90) [27]. They showed in both cases that, as the pixel dimensions of the patterns increased, the developmental methods outperformed the direct methods. It is noteworthy that performance disparity was much more marked for the relatively regular Norwegian flag pattern than for pattern generated by a 1D cellular automata. Hornby and Pollack evolved context free L-systems to define three dimensional objects (table designs) [16]. They found that their generative system could produce designs with higher fitness faster than direct methods. They point out that generative or developmental systems will scale better than direct methods when modularity is present. In furniture design, for instance, if there is a module that is responsible for producing a table leg, evolution only needs to alter and perfect one module rather than having to independently adjust an arbitrary number of independent table leg producing coding regions. A number of genotype-phenotype mappings on a problem of creating a tessellating tile pattern were examined in [4]. The authors found that an indirect developmental mapping (that they referred to as an implicit embryogeny) could evolve tiling patterns much more quickly than a variety of other representations (including direct) and further, that they could be subsequently grown into much larger sized patterns.

One drawback that they reported was that implicit embryogeny tended to produce the same types of pattern (i.e. of relatively low complexity). As we will see later our results support this finding. In these applications, it appears that development is satisfactory for low complexity problems, where there are many regularities - possibly regardless of scale. Like direct encodings, their behavior does deteriorate as the phenotype scales. One can speculate that this is due to the decrease in the genotype-phenotype correlation with an increase in complexity of the phenotype, which in turn reduces evolvability, as Lehre and Haddow found [18].

Heritable information can also be passed on through mechanisms other than the DNA, and this will affect evolvability and the developmental processes. Such information is the subject of the field of Epigenetics [25]. It includes the maternal influences described earlier and modifications to the genome caused by the mother's interaction with the environment. It appears that in addition to the standard base pair encoding in DNA, the genome also carries another code, the epigenetic code, attached to DNA, and that information provided by this code affects the expression of certain genes. Heritable epigenetic information alters the packing density of the DNA, changing the likelihood of genes being expressed. For example, research comparing the development of twins, shows that epigenetic codes may be more sensitive to environmental influences than DNA [25], where it is reported that just by making changes to the diet of a pregnant mouse, the coat color of pups can

be changed. The role of epigenetics in nature is still controversial, but there are a growing number of examples demonstrating how the environment alters gene expression.

In addition to the effects on development, epigenetics also may affect the evolvability of a species, as epigenetic information is somewhat heritable. Roemer et al. showed that manipulations of epigenetic information in mice were passed down to offspring [26], and comment that "If epigenetic inheritance indeed exists, what is its evolutionary significance? The extent of its effects will depend on the number of genetic loci in the genome that can be modified epigenetically, and on the stability of the modifications. Whether 'epimutations' have any adaptive significance also remains to be established. It should be emphasized that this type of inheritance is rooted firmly in Darwinian selection, with selection possibly both for the modified locus and for the genes that control epigenetic modifications." The use of maternal effects has been demonstrated in developmental neural networks. Matos et al. [19] found that the use of the maternal genotype decreased evolvability. They speculate that this may be due to lag from evolutionary momentum. In a second experiment, they looked at how placental interaction with the mother affected the evolvability of the neural networks. Here they found improvements over a standard developmental approach. It is clear that maternal influences shape developmental behavior, and developing suitable analogies may help artificial development.

9.5 The role of physics

Development in the real world is not just the product of genes. There are interactions with the environment, and in particular the limitations of biological chemistry and physics constrain what biological processes can do. It is likely that primitive life forms relied more on the properties of matter, such as viscoelasticity and chemical/mechanical excitation, rather than on gene expression. Forgacs and Newman argue that such physical properties are a "rough sort development", and that we should not expect either genes or physics to be sufficient on their own [10]. Earlier we discussed that the shapes of forming organs are the result of a small number of processes. These basic processes are determined by the physical properties of the cell namely adhesion, diffusion and viscoelasticity — which, incidentally, are also found in non-living systems and are not under any form of genetic control.

Early multicellular life consisted of cell aggregates. These cell aggregates would have the ability to self-organize into patterns, based on the chemical activity of each cell. Essentially, a cell aggregate would be an excitable medium. The primary role for cell adhesion would be for tissue formation. Due to their chemistries, biological cells have different rates of cell adhesion, which leads to an interesting property during early stages of development. Mature tissues have strong, long-lived links between the cells. Cell adhesion also allows for

ions and small molecules to pass between neighboring cells – without allowing other ions or molecules from the outside to get in.

During early development, however, cells are not joined in this way, but move easily as if in a liquid. This, combined with differential adhesion, forces the cells to become sorted as they move. Furthermore, some cells have adhesive polarity, which causes certain patterns to be formed, since the cells wish to reorganize themselves into a stable, low energy state. A striking example is when a mixture of cells (in this instance from the endodermal and ectodermal germ layers) from an organism called a hydra, are mixed together (producing a random pattern): They will sort themselves into the precise arrangement found in the original organism [29].

When cells evolved to have a variety of types, each with different adhesive properties, these effects of cell sorting occurred and new spatial patterns were constructed. As those properties were coupled with the evolution of polarized cells, the cells could form lumens or elastic sheets. In artificial development, we can use the properties of cell adhesion to generate some target patterns without having to evolve a gene regulatory network (or equivalent). To illustrate this, we present in the following section a method for evolving patterns, including the familiar French flag, using differential cell adhesion.

We would expect that using physical effects such as cell sorting would have limited utility on its own. However, just like in nature, a combination of inherent physical effects and control by a genotype might yield a high degree of sophistication. One advantage of a strong bias toward the physical control of development, compared to the genetic control, is that new cell formations can be achieved through minute genetic change. This may be very important for search algorithms, as one can explore the search space in unusual ways. In effect, a combination of both physics and genetics, and different ratios of the influence between these two factors may give a search heuristic that contains two very different algorithmic aspects. Perhaps as in nature, the balance between the responsibilities of each will be automatically optimized by evolution.

In artificial development the constraints of reality provided by physics do not exist. That means that their benefits, namely to guide and constrain search, are lacking. Because it appears that physics is useful in natural systems, one should perhaps find an analogous artificial physics for artificial development. At present, it is unclear what the artificial equivalents of cell adhesion, surface tension, gravity and diffusion are. It is also unclear what the relationship would be between developmental physics and the physics of the hardware on which the artificial developmental system is implemented. Work in evolvable hardware has shown that evolution is able to make use of the physical properties of its environment, and perhaps we can expect the same from development.

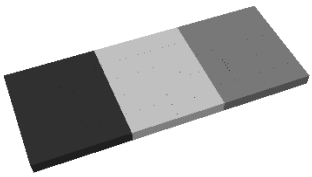

Fig. 9.3. Target French flag pattern.

9.6 Results from a Cell Sorting Experiment

Our model for cell growth consists of a grid whose points can either hold a cell or be empty. There are three different cell types (to map to the red, white and blue fields of the flags) and each of them has its own adhesive properties – to be determined by evolution. To simulate the flow of cells, we employ a simple mechanism whereby cells that wish to move can jump into a neighboring empty cell, or swap places with an existing neighboring cell. When the simulation is run, a cell is picked and a calculation is performed to see whether the entropy of the cell would drop if it were to swap with any of its neighboring cells, as described in [10, chapter 4]. The energy of a particular cell is calculated as the sum of the differences between the adhesion coefficients of the center cell and its neighbors. If a suitable swap is found, then cells swap position. This is repeated a number of times; the number of cell swaps allowed is determined by evolution. We also allow the cells to split, in order for the artificial embryo to grow. If there is an empty neighboring cell, then a cell can divide into this gap and take on the same cell type as its parent. The number of times cells are allowed to split is also determined by evolution.

For these experiments we tried two different approaches to the evolutionary system. In the first, we use two different chromosomes. In one, the chromosome specifies the cell adhesion properties of all cell types, the maximum number of swaps allowed and the maximum number of times cells are allowed to split. The other chromosome type contains a list of cells and their positions, in addition to the above properties. This list is used to define the starting configuration of the developing embryo. These initial cell positions may be analogous to the maternal influences described in section 9.2.

We also investigate the behavior without the evolved starting positions but with a scaled down version of the target, defined by hand. Here only the adhesion coefficients need to be evolved.

Adhesion coefficients are represented by floating point numbers. The cell positions are stored as a variable-length list of coordinates and cell types. In this model we ignore the possibility of cell polarization. Integers are used to store the number of iterations the simulation runs, and how many cell divisions are possible.

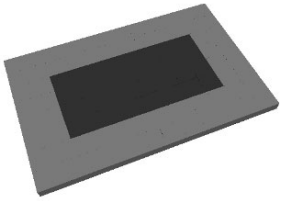

Fig. 9.4. Target cell cluster

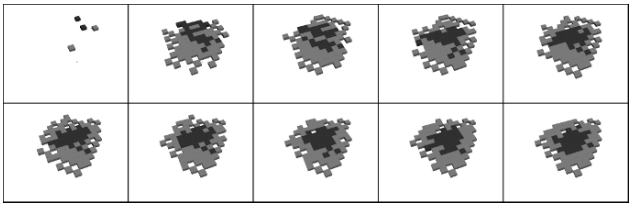

Fig. 9.5. Example of developing cell cluster, where the cell sorting moves the darker cells to the middle of the cell mass.

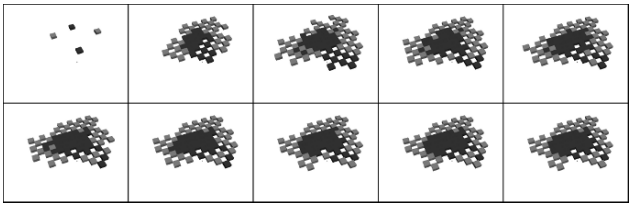

Fig. 9.6. A second example of a developing cell cluster. Again a central dark mass is formed, but here the outer cells produce a spaced pattern.

The fitness function determining the success of a solution counts the number of cells in resultant arrangements that were the same as in the target.

A basic evolutionary algorithm was used, consisting of a population of 50 individuals with tournament selection and elitism. In addition to mutation, we employ a basic two-point crossover on the genotype. Evolution was allowed to run for 5000 generations.

For these experiments, the target pattern was a cell cluster surrounded by an outer cell layer. Figure 9.4 shows the target image, which has similar form to some biological formations such as retinas ([10, page 93]).

Figures 9.5 and 9.6 shows two examples of evolved cell clusters, here evolution was allowed to determine the starting configuration for the cells (top left frame of each sequence). The behavior is similar to that created when the initial target pattern was not specified by the chromosome, such as in runs illustrated in the first ten frames of figure 9.7.

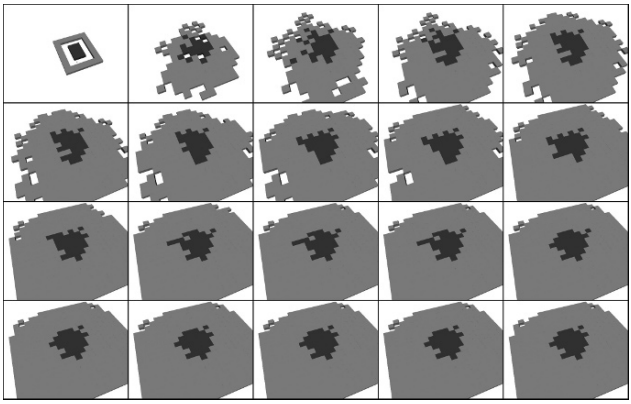

Fig. 9.7. Running the cell sorting beyond the period specified by evolution results in stable patterns. Here the first ten frames are from the period found during evolution to produce maturity. The following frames show the effect of running the simulation for additional time.

Figure 9.7 shows the effect of running a simulation for twice the period of time specified by evolution. The first 10 frames show the sequence that was used in fitness evaluation, and the following frames show what happens after the embryo has reached "maturity". We see that the general shape remains consistent, and that the center of the cell continues to adjust until it finds a point of minimum energy and stabilizes.

As in nature, embryos undergoing development are able to repair damage to some extent. Figure 9.8 shows the same developing embryo as figure 9.7, but this time the embryo is damaged by removing a band of cells. The embryo remains disrupted, however it starts to reform into the target pattern. This ability was not selected for during evolution, and is the result of the physics of development being used for this secondary purpose. Differential cell adhesion is not only responsible for sorting the cells into groups but also for bringing different cell clusters together. This is likely to be one of the mechanisms used for repair in the developing organism.

Other shapes can be produced by evolving the starting configurations, and allowing cell growth and cell sorting to finalize the pattern. For example, figure 9.9 shows an evolved French flag. In contrast to the previous French flag patterns discussed here, this one did not require the evolution of a program to control the behavior of the cells.

The final example is a complicated pattern, based on the types of behavior seen during this artificial development. Figure 9.10 shows a checkerboard in the shape of a triangle, attached to a basic two-color "flag". The purpose of these shapes is to demonstrate that we can evolve a variety of target patterns that have rounded shapes, solid layered masses, shapes containing patterns of empty space and regular structures with sharp edges. In nature there are a limited number of forms that cell groups can form, and these basic patterns

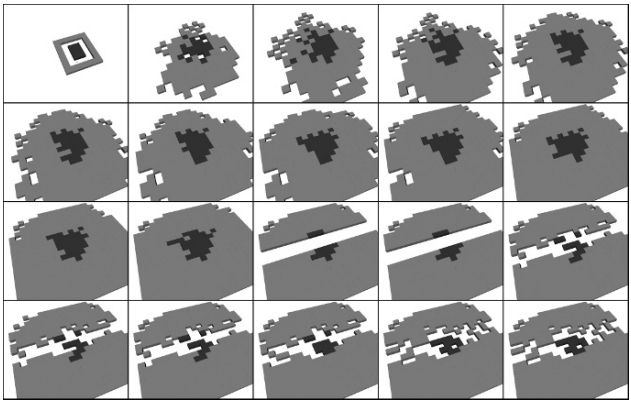

Fig. 9.8. An example of the regrowth of a damaged artificial embryo

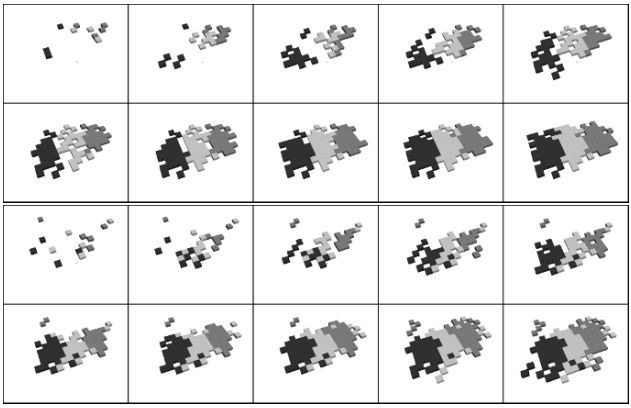

Fig. 9.9. Two examples of a French Flag produced by cell sorting.

are used as building blocks for organs. Without differential cell adhesion we found it impossible to get checkerboard patterns to evolve. The nearest that could be obtained were three groups of cells with a large amount of mixing. This demonstrates that cell movement can be a useful and important part of developmental systems.

Figure 9.11 shows an interesting result observed during testing of the simulation software. Here, each cell group has slightly different cell adhesion properties and the initial state is a randomized cell cluster. However, without any guidance from evolution, a rough French flag pattern is produced. As in nature, certain patterns are perhaps an inevitable consequence of the physical properties of the cells that make them. If this is true, it is important to understand to what extent the patterns are restricted by biological development and to draw conclusions for artificial development.

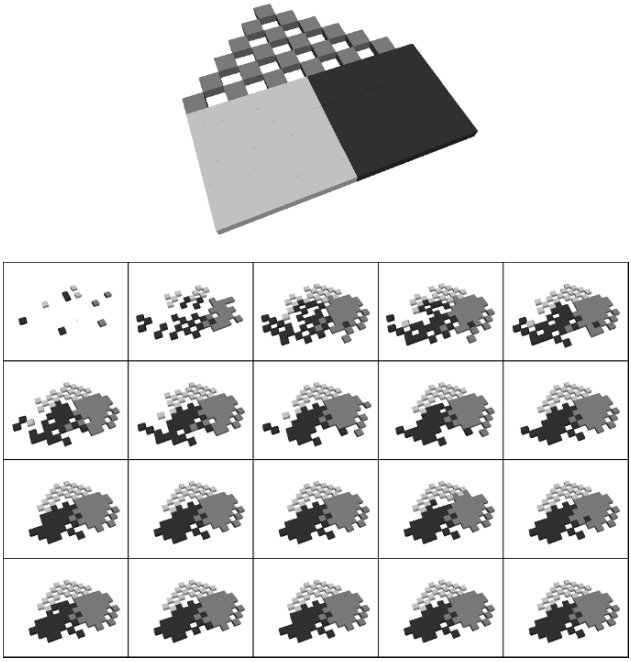

Fig. 9.10. Target checker board and flag, and an example of a developing embryo that forms this pattern.

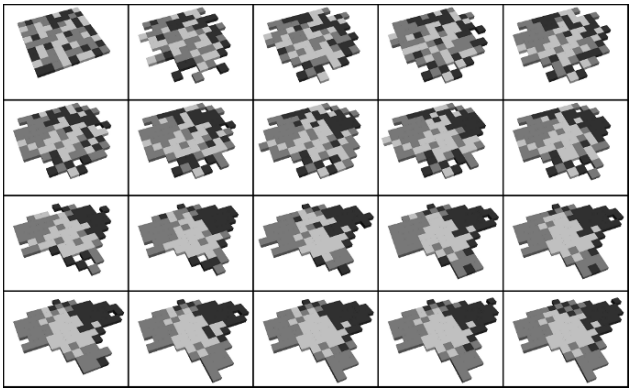

Fig. 9.11. French flag emerging because of differential cell adhesion

9.7 Conclusions

Developmental systems may prove to be a very useful technique in computer science. The field is, however, still in its infancy, and it is difficult to see how the developmental analogy can be applied to many of the typical problems in computer science. Specifically, the challenge is to map development into a computational domain. The applications described here have demonstrated its utility in producing patterns, whether as abstract images or as topographies for neural networks, but transforming these preliminary ideas into a more generalized and practical computational system remains to be done. Downing argues that this is "largely because embryogenesis evolved for the purpose of synthesizing 3-dimensional structure from a linear code, not for growing Universal Turing Machines" [6], and that while we can map problems onto a developmental framework, it is unclear whether this is an inherently suitable approach. Despite these issues, development has many features that are attractive in artificial systems — and if we can get these ideas to work, we will have another powerful, bio-inspired technique to apply.

References

1. W. Banzhaf and C. Lasarczyk. Genetic programming of an algorithmic chemistry. *Genetic Programming Theory and Practice II*, 8:175–190, 2004.
2. W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming: An Introduction.* Morgan Kaufmann, San Francisco, 1998.
3. P. Bentley. Investigations into graceful degradation of evolutionary developmental software. *Natural Computing*, 4(4):417–437, 2005.
4. P. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 35–43, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.
5. A. Carrascal, D. Manrique, D. Pérez, J. Ríos, and C. Rossi. Growing axons evolving l-systems. In M. Hamza, editor, *proceedings Artificial Intelligence and Applications*, volume 403. Acta Press, 2003.
6. K. L. Downing. Developmental models for emergent computation. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *International Conference on Evolvable Systems(ICES)*, volume 2606 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2003.
7. A. Eiben and J. Smith. *Introduction to Evolutionary Computing.* Springer, Berlin, 2003.
8. G. Escuela, G. Ochoa, and N. Krasnogor. Evolving L-systems to capture protein structure native conformations. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 74–84, Lausanne, Switzerland, 30 Mar. - 1 Apr. 2005. Springer.
9. D. Federici. Evolving developing spiking neural networks. In *proceedings of CEC 2005 IEEE Congress on Evolutionary Computation*, pages 543– 550, 2005.

10. G. Forgacs and S. A. Newman. *Biological Physics Of The Developing Embryo.* Cambridge University Press, 2005.
11. J. Gerhart and M. Kirschner. *Cells, embryos, and evolution : toward a cellular and developmental understanding of phenotypic variation and evolutionary adaptability.* Malden, Mass. : Blackwell Science, 1997.
12. F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon, France, 1994.
13. F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Stanford University, CA, USA, 28–31 1996. MIT Press.
14. S. Harding. *Evolution In Materio.* PhD thesis, University of York, 2005.
15. S. Harding and J. F. Miller. The dead state: A comparison between developmental and direct encodings. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2006.
16. G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 600–607, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.
17. H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–467, 1990.
18. P. Lehre and P. Haddow. Developmental mappings and phenotypic complexity. In *proceedings of IEEE Congress on Evolutionary Computation(CEC) 2003*, pages 62–68, 2003.
19. A. Matos, R. Suzuki, and T. Arita. Evolutionary models for maternal effects in simulated developmental systems. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 149–150, New York, NY, USA, 2005. ACM Press.
20. J. Miller and M. Hartmann. Untidy evolution: Evolving messy gates for fault tolerance. In *Proceedings of The 4th International Conference on Evolvable Systems: From Biology to Hardware, ICES2001*, volume 2210 of *Lecture notes in computer science*, pages 14–25, Tokyo, Japan, 2001. Springer-Verlag.
21. J. F. Miller. Evolving a self-repairing, self-regulating, french flag organism. In 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, *GECCO (1)*, volume 3102 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 2004.
22. M. Mitchell. *An introduction to genetic algorithms.* MIT Press, Cambridge, MA, USA, 1996.
23. C. Nüsslein-Volhard. *Coming To Life.* W. W. Norton and Company, 2006.
24. C. Ortega-Sanchez, D. Mange, S. Smith, and A. Tyrrell. Embryonics: A bio-inspired cellular architecture with fault-tolerant properties. *Genetic Programming and Evolvable Machines*, 1(3):187–215, 2000.
25. J. Qiu. Epigenetics: unfinished symphony. *Nature*, 441:143–145, May 2006.
26. I. Roemer, W. Reik, W. Dean, and J. Klose. Epigenetic inheritance in the mouse. *Current Biology*, 7:277–280, 1997.

27. D. Roggen and D. Federici. Multi-cellular development: is there scalability and robustness to gain? In X. Yao, E. Burke, and J. L. et al., editors, *proceedings of Parallel Problem Solving from Nature 8, Parallel Problem Solving from Nature (PPSN) 2004*, pages 391–400, 2004.
28. A. Siddiqi and S. Lucas. A comparison of matrix rewriting versus direct encoding for evolving neural networks. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, (Piscataway, NJ, USA)*, pages 392–397. IEEE Press, 1998.
29. U. Technau and T. Holstein. Cell sorting during the regeneration of hydra from reaggregated cells. *Developmental biology*, 151(1):117–27, 1992.
30. A. Thompson and C. Wasshuber. Design of single electron systems through artificial evolution. In *Int. J. Circuit Theory and Applications*, volume 28 (6), pages 585–599, 2000.