

Evolving Fractal Gene Regulatory Networks for Robot Control

Peter J. Bentley

Department of Computer Science, University College London, Gower Street, London.
P.Bentley@cs.ucl.ac.uk

Abstract. Fractal proteins are a new evolvable method of mapping genotype to phenotype through a developmental process, where genes are expressed into proteins comprised of subsets of the Mandelbrot Set. The resulting network of gene and protein interactions can be designed by evolution to produce specific patterns, that in turn can be used to solve problems. Here the use of fractal gene regulatory networks for learning a robot path through a series of obstacles is described. The results indicate the ability of this system to learn regularities in solutions and automatically create and use modules.

1 Introduction

Life is complex. This is true in the chemical interactions of proteins and genes within a single cell, or in the cellular interactions in a multicellular organism. While evolution is mostly to blame for this, there can be little doubt that complexity would not arise if the vast intricacies of molecular interactions and physical forces were not present. Open-ended evolution (evolution in which solutions become progressively more complex) relies on the right kind of genetic representation in the right kind of environment. In nature, this is DNA – a molecule that relies on chemical interactions in order to function.

Translating these ideas into computer science is not a simple prospect. As we know in evolutionary computation, using a fixed binary string as a genotype, prohibits complexity growth. But variable-length representations such as genetic programming do not guarantee an increase in complexity either (unless you count introns as complexity). Even if a seemingly ideal representation is found, often it is not evolvable, or it only achieves its complexity increases through the careful high-level structures (e.g. modularity) imposed on it by the developer.

This work takes a different approach. A developmental process maps variable-length genotypes to phenotypes, through the use of *fractal proteins*. Genes are expressed into complex fractal shapes (subsets of the Mandelbrot Set) that interact according to their forms. The resulting network of gene interactions can be designed by evolution to produce specific gene activation patterns, that in turn can be used to solve problems. In this paper the use of fractal gene regulatory networks for learning a robot path through a series of obstacles is described.

2 Background

Researchers such as Hornby [7], Bongard [5], Haddow [6] and Kumar and Bentley [10] have demonstrated that various types of development can enable smaller genotypes to represent more complex phenotypes through the ability of development to discover modularities and repetition. Other scientists in the field have been focusing on the ability of developmental methods to enable self-repairing behaviour and graceful degradation of solutions. For example, the work of Andy Tyrrell and his group create fault-tolerant hardware inspired by ideas of embryology and immune systems [8]. More recently, Julian Miller has described experiments evolving developmental programs to create “French Flag” patterns [12]. He shows that development is able to regenerate these patterns. However, work on applying developmental algorithms to robot control is less common. Most seem to rely on the development of neural networks that are then used to control motion (and in some cases generate form) [9] [7] [5].

3 Fractal Proteins

Development is the set of processes that lead from egg to embryo to adult. Instead of using a gene for a parameter value as we do in standard EC (i.e., a gene for long legs), natural development uses genes to define proteins. If expressed, every gene generates a specific protein. This protein might activate or suppress other genes, might be used for signalling amongst other cells, or might modify the function of the cell it lies within. The result is an emergent “computer program” made from dynamically forming gene regulatory networks (GRNs) that control all cell growth, position and behaviour in a developing creature [13].

Table 1. Types of objects in the representation

<i>fractal proteins</i>	defined as subsets of the Mandelbrot set.
<i>Environment</i>	contains one or more fractal proteins (expressed from the environment gene(s)), and one or more <i>cells</i> .
<i>Cell</i>	contains a <i>genome</i> and <i>cytoplasm</i> , and has some <i>behaviours</i> .
<i>Cytoplasm</i>	contains one or more fractal proteins.
<i>Genome</i>	comprising <i>structural genes</i> and <i>regulatory genes</i> . In this work, the structural genes are divided into different types: <i>cell receptor genes</i> , <i>environment genes</i> and <i>behavioural genes</i> .
<i>regulatory gene</i>	comprising operator (or promoter) region and coding (or output) region.
<i>cell receptor gene</i>	a structural gene with a coding region which acts like a mask, permitting variable portions of the environmental proteins to enter the corresponding cell cytoplasm.
<i>environment gene</i>	a structural gene which determines which proteins (maternal factors) will be present in the environment of the cell(s).
<i>behavioural gene</i>	structural gene comprising operator and cellular behaviour region.

In this work, a biologically plausible model of gene regulatory networks is constructed through the use of genes that are expressed into *fractal proteins* – subsets of the Mandelbrot set that can interact and react according to their own fractal chemistry. Further motivations and discussions on fractal proteins are provided in [1][2][3]. Table 1 describes the object types in the representation; Figure 1 illustrates the representation. Figure 2 provides an overview of the algorithm used to develop a phenotype from a genotype. Note how most of the dynamics rely on the interaction of fractal proteins. Evolution is used to design genes that are expressed into fractal proteins with specific shapes, which result in developmental processes with specific dynamics.

FRACTAL DEVELOPMENT

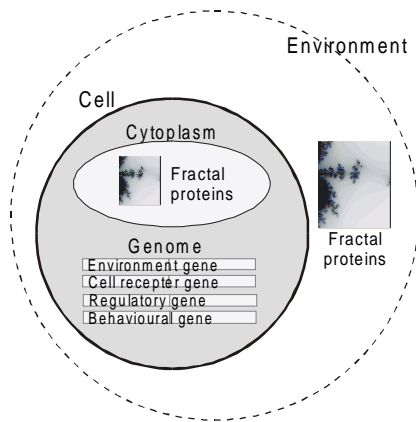


Fig. 1. Representation using fractal proteins.

For every developmental time step:

For every cell in the embryo:

Express all environment genes and calculate shape of merged environment fractal proteins

Express cell receptor genes as receptor fractal proteins and use each one to mask the merged environment proteins into the cell cytoplasm.

If the merged contents of the cytoplasm match a promoter of a regulatory gene, express the coding region of the gene, adding the resultant fractal protein to the cytoplasm.

If the merged contents of the cytoplasm match a promoter of a behavioural gene, use coding region of the gene to specify a cellular function.

Update the concentration levels of all proteins in the cytoplasm. If the concentration level of a protein falls to zero, that protein does not exist.

Fig. 2. The fractal development algorithm.

3.2 Defining a Fractal Protein

A fractal protein is a finite square subset of the Mandelbrot set [11], defined by three codons (x,y,z) that form the coding region of a gene in the genome of a cell. Each (x, y, z) triplet is expressed as a protein by calculating the square fractal subset with centre coordinates (x,y) and sides of length z , see fig. 3 for an example. In this way, it is possible to achieve as much complexity (or more) compared to natural protein folding in nature.

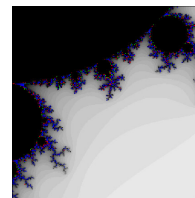


Fig. 3. Example of a fractal protein defined by $(x = 0.132541887, y = 0.698126164, z = 0.468306528)$

In addition to shape, each fractal protein represents a certain *concentration* of protein (from 0 meaning “does not exist” to 200 meaning “saturated”), determined

by protein production and diffusion rates [1].

3.3 Fractal Chemistry

Cell cytoplasm and the environment usually contain more than one fractal protein. In an attempt to harness the complexity available from these fractals, multiple proteins are merged. The result is a product of their own “fractal chemistry” which naturally emerges through the fractal interactions.

Fractal proteins are merged (for each point sampled) by iterating through the fractal equation of all proteins in “parallel”, and stopping as soon as the length of any is unbounded (i.e. greater than 2). Intuitively, this results in black regions being treated as though they are transparent, and paler regions “winning” over darker regions. See fig 4 for an example.

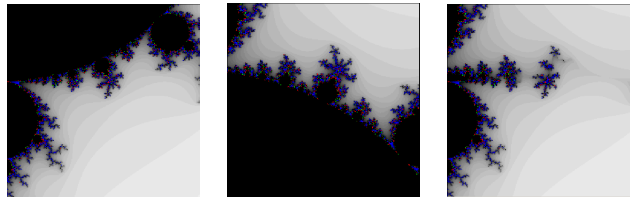


Fig. 4. Two fractal proteins (left and middle) and the resulting merged fractal protein combination (right).

3.4 Genes

The environment gene, cell receptor gene, regulatory genes, and behavioural genes all contain 7 real-coded values:

xp	yp	zp	<i>Affinity threshold</i>	<i>Concentration threshold</i>	x	y	z	<i>type</i>
------	------	------	---------------------------	--------------------------------	-----	-----	-----	-------------

where $(xp, yp, zp, Affinity\ threshold, Concentration\ threshold)$ defines the promoter (operator or precondition) for the gene and (x,y,z) defines the coding region of the gene. The *type* value defines which type of gene is being represented, and can be one or all of the following: *environment*, *receptor*, *behavioural*, or *regulatory*. This enables the type of genes to be set independently of their position in the genome, enabling variable-length genomes. It also enables genes to be multi-functional, i.e. a gene might be expressed both as an environmental protein and a behaviour.

When *Affinity threshold* is a positive value, one or more proteins must match the promoter shape defined by (xp,yp,zp) with a difference equal to or lower than *Affinity threshold* for the gene to be activated. When *Affinity threshold* is a negative value, one or more proteins must match the promoter shape defined by (xp,yp,zp)

with a difference equal to or lower than $|Affinity\ threshold|$ for the gene to be repressed (not activated).

To calculate whether a gene should be activated, all fractal proteins in the cell cytoplasm are merged (including the masked environmental proteins) and the combined fractal mixture is compared to the promoter region of the gene. The full details of this process are lengthy; interested readers should consult [1][2][3].

Behavioural Gene. A behavioural gene is activated when other protein(s) in the cytoplasm match its promoter region (using the *affinity threshold*). For this application, a gradual activation between ‘not activated’ and ‘activated’ was required, using the x value of the coding region (x,y,z) triplet as a *fate* value to define a function, calculated as follows:

If the gene is being activated with a negative *Affinity threshold*,
 $output = output \cdot (totalconcentration \cdot concentrationthreshold) \times fate$

If the gene is being activated with a positive *Affinity threshold*,
 $output = output + (totalconcentration \cdot concentrationthreshold) \times fate$

Note how the total concentration of proteins seen on the promoter is offset against the *Concentration Threshold* gene and scaled by the *fate* gene (x value of the coding region), allowing evolution to adjust the range of values seen on the output, and used to specify behaviours. (If there are more behavioural genes than are required, the resultant behaviour will be the sum behaviour of all genes.)

3.7 Development

As was illustrated in figure 2, an individual begins life as a single cell in a given environment. To develop the individual from this zygote into the final phenotype, fractal proteins are iteratively calculated and matched against all genes of the genome. Should any genes be activated, the result of their activation (be it a new protein, receptor or cellular behaviour) is generated at the end of the current cycle. Development continues for d cycles, where d is dependent on the problem. Note that if one of the cellular behaviours includes the creation of new cells, then development will iterate through all genes of the genome in all cells.

3.8 Evolution

The genetic algorithm used in this work has been used extensively elsewhere for other applications (including GADES [4]). A dual population structure is employed, where child solutions are maintained and evaluated, and then inserted into a larger adult population, replacing the least fit. The fittest n are randomly picked as parents from the adult population. Typically the child population size is set to 80% of the

adult size and $n = 40\%$. (For further details of this GA, refer to [4].) Because real coding was used, duplication and creep mutation is used, see [1] for complete details. Crossover is always applied; all mutations occur with probability 0.01 per gene.

4 Robot Control

Previous work has demonstrated how evolution can generate specific fractal proteins that interact with each other in order to produce desired patterns of behavioural gene activation [1][2][3]. Here the two behavioural genes are used to generate commands for a robot, with the aim of directing the robot past obstacles in its environment to reach a destination. So instead of directing cellular behaviour (i.e., cell division, differentiation or death), the fractal gene regulatory networks will direct robot behaviour. Although the tasks are clearly very different, both do rely on precise timing and accuracy, and both make use of the inherent pattern-generating properties of GRNs.

4.1 Robot

The platform used was a palm-sized six-legged robot known as a *wonderborg*TM, produced by Bandai, see Fig. 5. Although not widely available outside Japan, the robot boasts several useful features: it is programmed via infrared communications, and once programmed it runs fully autonomously. It has two forward-facing infrared collision detectors, a bottom-facing infrared floor sensor, an upwards facing light sensor and two microswitch touch sensors or feelers. Its legs are driven by two motors (one for the legs on each side), operating like a tracked vehicle.

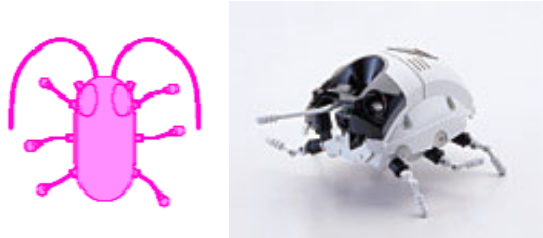











Fig. 5. The *wonderborg*TM robot (Bandai).

The *wonderborg*TM was not designed as a research platform, so unfortunately suffers from certain drawbacks. Currently the only way to send a program to the robot is through Bandai's proprietary software which forces the use of high-level commands and programming structures. Because all movement commands are high-level (e.g. "Rotate right" or "Back up left"), this prohibits the independent control of motors. To overcome the difficulties, the two behavioural genes were treated as "steering" and "acceleration," and then their values were mapped onto the appropriate high-

level commands. For example, a positive value for the steering gene is taken to mean “steer right” and a value of -3 for the acceleration gene is taken to mean “move -3 backwards”, see table 2. The fractal development system was then modified to convert the patterns of gene activation produced during development into a robot control file in the appropriate format for the proprietary software.

Table 2. Mapping of steering and acceleration genes to the nine predefined robot commands

Steering	Acceleration	Robot command	Symbol ($n=1$)
0 (None)	0 (None)	<i>Halt</i>	
-ve (Left)	0 (None)	<i>Rotate left</i>	
+ve (Right)	0 (None)	<i>Rotate right</i>	
0 (None)	n (n forwards)	<i>Advance by n</i>	
-ve (Left)	n (n forwards)	<i>Turn left by n</i>	
+ve (Right)	n (n forwards)	<i>Turn right by n</i>	
0 (None)	$-n$ (n backwards)	<i>Back up by n</i>	
-ve (Left)	$-n$ (n backwards)	<i>Back up left by n</i>	
+ve (Right)	$-n$ (n backwards)	<i>Back up right by n</i>	

Finally, to enable high-speed evaluation of robot control programs, a *wonderborg*TM simulator was created. This reads the same file format as used by Bandai’s proprietary software (and as output by the fractal developmental system) and calculates the path of the robot through an environment. The simulator was designed to be fast – on a 1 Ghz PC, approximately 40 developmental cycles and corresponding robot control simulations occur every second.

5 Experiments

Two sets of experiments were performed. The first used a simple environment with four obstacles, the second used a more difficult environment with seven obstacles, see figures 6 and 7. The robot simulator was initialised with the robot at one end of the environment. The further the robot managed to walk across the environment the higher the fitness of the corresponding controller. If the robot touched an obstacle or walked off the edges, its final position was measured as its last valid position in the environment. Note that the controllers did not use any of the robot’s sensors to gauge proximity of obstacles (which might have made the task easier). Instead they specified a fixed path past the obstacles, learning the structure of the environment through generations of “hitting its head against walls.” A second fitness measure (of less importance than the first) was used to provide penalties corresponding to the time taken by the robot and hence encourage efficient and fast journeys. (Without this, the robots would often rotate many times on one spot for several seconds between each movement in order to find the perfect angle.)

To evolve the controllers, the fractal development system was initialised with a single cell, 1 environment gene, 1 receptor gene, 2 behavioural genes and 6 regulatory genes. (Note that with variable length genomes, evolution was free to modify

these gene numbers). The operator and coding regions of the genes were randomly initialised with the alleles that defined 10 previously evolved protein fractals [1]. 32 developmental steps were employed, and the evolutionary algorithm used a population size of 100, running for up to 500 generations in the first experiment and 1500 in the second.

6 Results and Analysis

In the first experiment, 13 out of 20 produced robot controllers able to reach the top (500 generations). In the second experiment, only 1 out of 20 produced successful robot controllers after 500 generations. When the GA was allowed to run for 1500 generations, 11 out of 20 produced robot controllers able to reach the top. The increase in success is further evidence of evolvability of this representation [1][3]; it is likely that further generations would have improved the success rate even more. Figs 6 and 7 show examples of robot paths produced by fit controllers. The final controllers have been confirmed by testing on the actual robot.

Despite the poor level of control available through the high-level commands, evolution and development manage to create remarkably precise and detailed robot paths, designed to avoid the obstacles in the environment. Significantly, these controllers often employed “modules” or “subroutines” that were repeated (sometimes with variations) to overcome the obstacles. This automatic discovery and exploitation of pattern in the solution is very important, and also very unusual – most other systems (e.g. ADFs in GP or parameterised L-Systems) must explicitly build in notions of modules. Here they emerge naturally. To illustrate a little of how this happens, figure 8 displays protein concentrations of the controller in fig 7c.

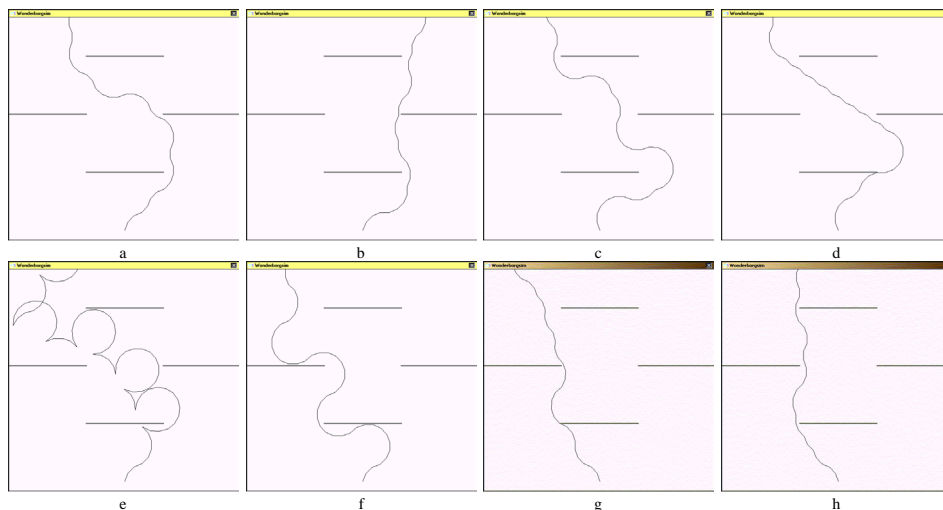


Fig. 6. Eight examples of very fit controllers for the first experiment.

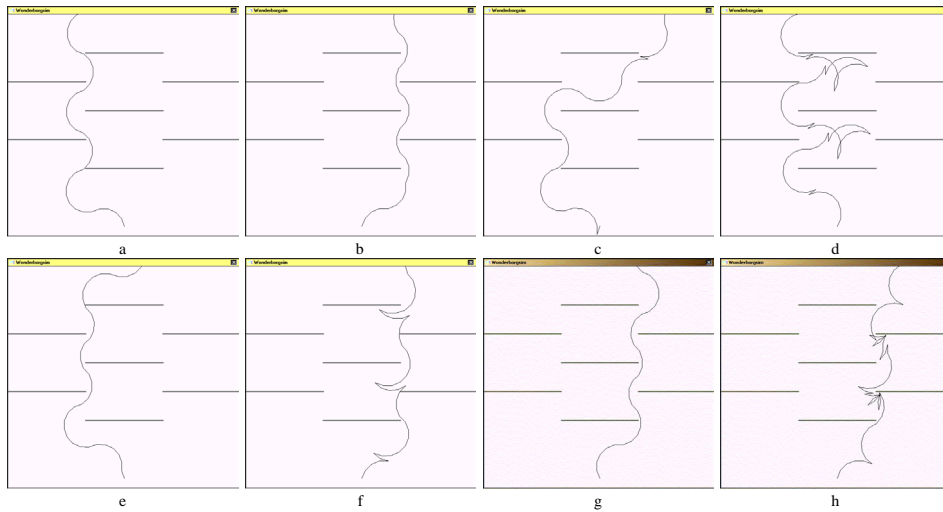


Fig. 7. Eight examples of very fit controllers for the second experiment.

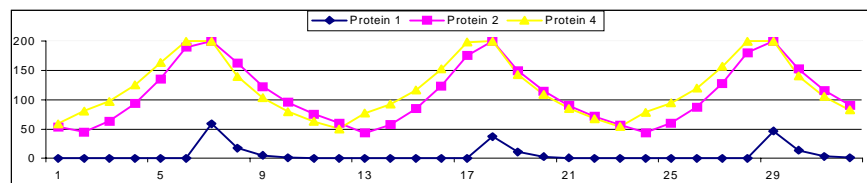



Fig. 8. Protein concentrations of the three main proteins used during development in controller shown in fig 7c. Steering and acceleration are the result of interactions between oscillating protein concentrations, resulting in very precisely controlled oscillating robot motion. The 32-step pattern is partially repeated in the final controller, producing a total of 50 commands for the robot.

7 Conclusions

It is not a trivial task to find a suitable genetic representation that enables open-ended evolution, while being evolvable, incorporating ideas of intricate chemical/physical environments, and employing developmental processes. The use of genes expressed as fractal proteins is the approach investigated in this paper. The work has shown that fractal gene regulatory networks can be successfully designed by evolution to solve problems. Here, the task of producing robot controllers capable of guiding a “bug” robot past obstacles in an environment was demonstrated. In addition to creating diverse and useful controllers, the fractal GRN also demonstrated an ability to identify patterns in solutions and create its own modules (sub-solutions that were reused and used with minor modifications). This automatic learning, not just of a good solution (robot path), but of a way of *building* a good solution in an efficient manner, is seen as a significant and important property of developmental processes. Combine this with the results of previous research that

shows a tendency towards efficiency and fault-tolerance of fractal GRNs [3], and the potential for this technique looks impressive.

Acknowledgments

Thanks to Sanjeev Kumar for his comments. This material is based upon work supported by the European Office of Aerospace Research and Development (EOARD), Airforce Office of Scientific Research, Airforce Research Laboratory, under Contract No. F61775-02-WE014. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of EOARD. MOBIUS is an  project <http://www.cs.ucl.ac.uk/staff/P.Bentley/emblem.html>

References

1. Bentley, P. J. Fractal Proteins. 2003a. To appear in Genetic Programming and Evolvable Machines Journal.
2. Bentley, P. J. Evolving Fractal Proteins. 2003b. In Proc. of ICES '03, the 5th International Conference on Evolvable Systems: From Biology to Hardware.
3. Bentley, P. J. Evolving Beyond Perfection: An Investigation of the Effects of Long-Term Evolution on Fractal Gene Regulatory Networks.2003c. Submitted to *Information Processing in Cells and Tissues* (IPCAT 2003).
4. Bentley, P. J. From Coffee Tables to Hospitals: Generic Evolutionary Design. 1999. Chapter 18 in Bentley, P. J. (Ed) *Evolutionary Design by Computers*. Morgan Kaufmann Pub. San Francisco, pp. 405-423.
5. Bongard, J. C. Evolving Modular Genetic Regulatory Networks. 2002. In *Proc. of 2002 Congress on Evolutionary Computation (CEC2002)*, IEEE Press, pp. 1872-1877.
6. P. C. Haddow, G. Tuft, and P. van Remortel. Shrinking the Genotype: L-Systems for EHW 2001. In Proc. Of 4th Int. Conf. On Evolvable Systems: From Biology to Hardware, Tokyo, Japan.
7. Hornby, G. S. Generative Representations for Evolutionary Design Automation. 2003. Brandeis University, Dept. of Computer Science, Ph.D. Dissertation.
8. A.H. Jackson, A.M. Tyrrell Implementing Asynchronous Embryonic Circuits using AARDVArc. 2002. In *Proceedings of 2002 NASA/DoD Conference on Evolvable Hardware* (EH-2002), IEEE Computing Society, Alexandria, Virginia, Pages 231-240, 15-18.
9. N. Jakobi. Harnessing Morphogenesis. 1995. *International Conference on Information Processing in Cells and Tissues*, Liverpool, UK.
10. S. Kumar and P. J. Bentley. Computational Embryology: Past, Present and Future. 2003. Invited chapter in Ghosh and Tsutsui (Eds) *Theory and Application of Evolutionary Computation: Recent Trends*. Springer Verlag (UK).
11. Mandelbrot, B. *The Fractal Geometry of Nature*. 1982. W.H. Freeman & Company.
12. Miller, J. and Banzhaf, W. Evolving the Program for a Cell: From French Flags to Boolean Circuits. 2003. To appear as an invited chapter in Kumar, S. and Bentley, P. J. (Eds) *On Growth, Form and Computers*. Academic Press, 2003.
13. Lewis Wolpert, Rosa Beddington, Thomas Jessell, Peter Lawrence, Elliot Meyerowitz, Jim Smith. *Principles of Development, 2nd Ed.* 2001. Oxford University Press.