A 'Hands on' Strategy for Teaching Genetic Algorithms to Undergraduates

Anne Venables and Grace Tan
Victoria University, Melbourne City, Australia

Anne.Venables@vu.edu.au Grace.Tan@vu.edu.au

Executive Summary

Genetic algorithms (GAs) are a problem solving strategy that uses stochastic search. Since their introduction (Holland, 1975), GAs have proven to be particularly useful for solving problems that are 'intractable' using classical methods. The language of genetic algorithms (GAs) is heavily laced with biological metaphors from evolutionary literature, such as population, chromosome, crossover, cloning, mutation, genes and generations. For beginners studying genetic algorithms, there is quite an overhead in gaining comfort with these terms and an understanding of their parallel meanings in the unfamiliar computing milieu of an evolutionary algorithm.

This paper describes a 'hands on' strategy to introduce and teach genetic algorithms to undergraduate computing students. By borrowing an analogical model from senior biology classes, poppet beads are used to represent individuals in a population (Harrison, 2001). Described are several introductory exercises that transport students from an illustration of natural selection in *Biston betula* moths, onto the representation and solution of differing mathematical and computing problems. Through student manipulation and interactions with poppet beads, the exercises cover terms such as population, generation, chromosome, gene, mutation and crossover in both their biological and computing contexts. Importantly, the tasks underline the two key design issues of genetic algorithms: the choice of an appropriate chromosome representation, and a suitable fitness function for each specific instance. Finally, students are introduced to the notion of schema upon which genetic algorithms operate.

The constructivist model of learning advocates the use of such contextual problems to create an environment where students become active participants in their own learning (Ben-Ari, 1998; Greening, 2000; Kolb, 1984). Initial student feedback about these "hands on" exercises has been enthusiastic. As well, several students have made reference to the lessons learnt as the basis for GA coding in subsequent open-ended assignments. It seems that once the hurdle of becoming familiar with GA terminology has been surmounted, students find genetic algorithms to be particularly intriguing for their uncanny ability to solve incredibly complex problems quickly and proficiently (Moore, 2001).

Keywords: Genetic Algorithms, Experiential learning, Analogical model, Constructivism

Editor: Grandon Gill

Introduction

The study of data structures and their algorithms is fundamental in most undergraduate information technology and computer science programs and this can be simply evidenced by nearly 42,000 Google hits on the term “data structures course”. Renown for his seminal works in the field, Donald Knuth (Softpanorama, 2006) is credited with saying *"Languages come and go, but algorithms stand the test of time".* Others, like Alan Kay (2006), argue that many of the ideas that are presented in computing have become dated and that *“almost nothing exciting about computing today has to do with data structures and algorithms”.* We suggest that by introducing genetic algorithms (GAs) in our undergraduate computing programs, it is possible to still pay reverence to classical problems and their traditional algorithmic solutions whilst capturing the imagination of our students.

Genetic algorithms are a problem solving strategy that uses stochastic search. Since their introduction in 1975 (Holland, 1975), GAs have proven to be particularly useful for solving problems that are “intractable” using classical methods. In such situations, the problem is viewed as a landscape upon which possible solutions may be found. As seen in Figure 1, an individual solution is located and represented by its coordinates. For optimisation problems, where one is looking for the highest hill or the lowest valley, it can be seen that some individual solutions are better located than others. Borrowing heavily from the language of biological evolution, these solutions/individuals are said to be the fittest in a population of competing solutions. Then over successive iterations, the GA evolves the population of competing solutions until it becomes a population of consistently very fit individuals, as seen in Figure 2.

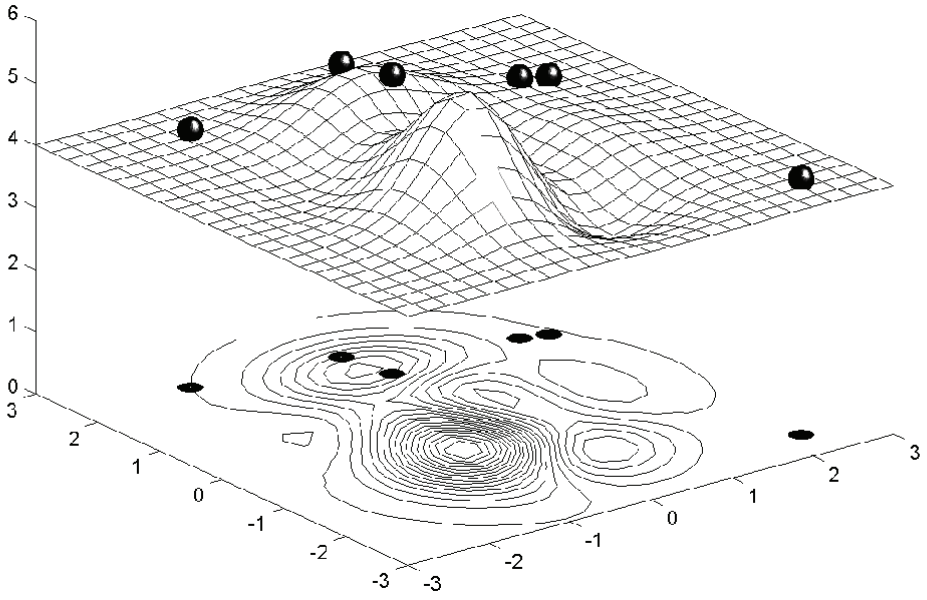

Figure 1: Landscape of the mathematical equation $f(x,y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x - x^3 - y^3) e^{-x^2-y^2}$ and six possible solutions (Negnevitsky, 2005).

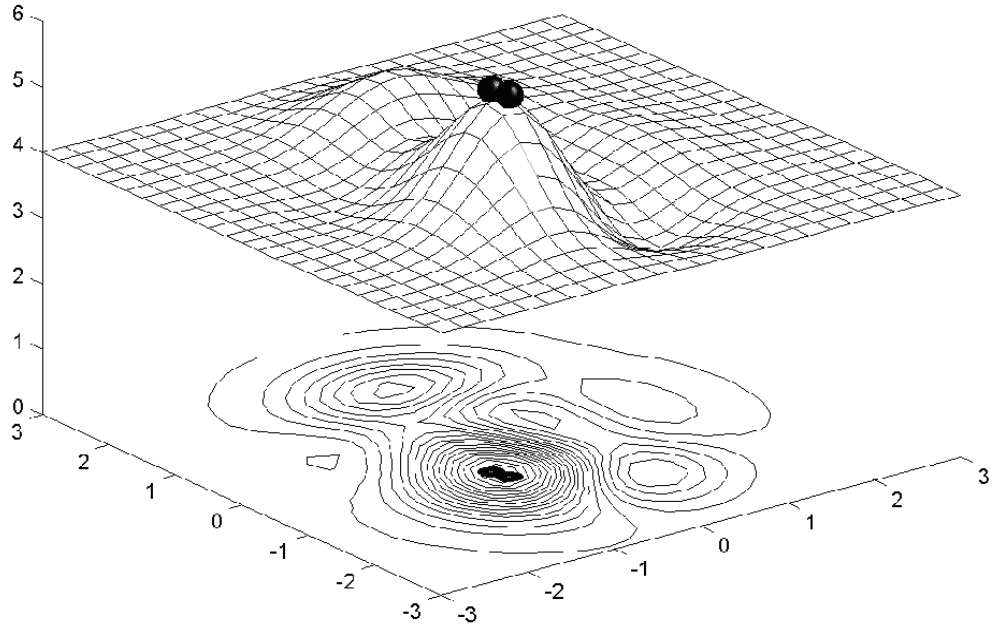

Figure 2: Landscape of the mathematical equation $f(x,y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x - x^3 - y^3)\, e^{-x^2-y^2}$ and six 'fitter' solutions (Negnevitsky, 2005).

To better describe the process that is a genetic algorithm, one relies even more heavily upon the biological metaphor. In a GA, each possible solution is coded using a data structure known as a chromosome. A chromosome is composed of a string of genes, each gene representing a specific input variable. Collectively the genes are used to evaluate the fitness of an individual solution. Then, proportional to their specific fitness value, chromosomes are allowed to reproduce using the genetic operators of crossover and mutation. So the fittest chromosomes are more likely to reproduce and contribute to the next generation of chromosomes whereas less fit individuals eventually become lost to future generations. Over successive iterations, the average fitness of the whole population improves and the genetic algorithm can be expected to breed an optimal solution to the problem (Negnevitsky, 2005).

For beginners with little or no exposure to the language of evolutionary literature, there is a considerable overhead of understanding genetic terminology and the underlying biological concepts as applied in a GA. However, if the hurdle of becoming familiar with the language can be surmounted, students find genetic algorithms to be particularly intriguing for their uncanny ability to solve incredibly complex problems quickly and proficiently (Moore, 2001). This paper outlines a strategy to quickly overcome student unfamiliarity with the biological concepts and language used in genetic algorithms. Perhaps not surprisingly, the approach has been borrowed from introductory genetics courses and it has undergone modification as to situate it more naturally for computing students.

The “Hands On” Strategy

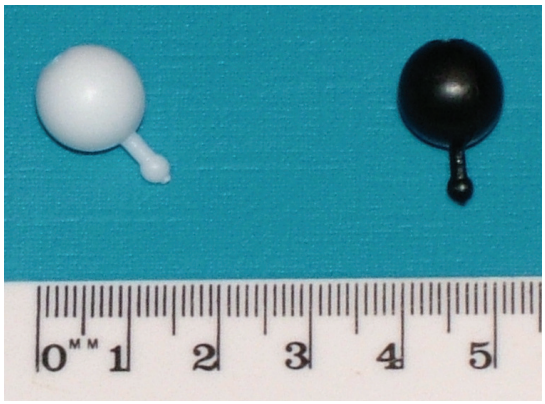

Figure 3: Examples of poppet beads

To introduce genetic algorithms to computing students, an analogical model is used. As described by Harrison (2001), an analogical model is one that helps *“people visualise the objects and processes which they are trying to understand”* and these models typically use *“a familiar object or experience to inform the learner about new and poorly understood objects, processes or concepts.”* In his research into the use of models for teaching difficult science concepts to teenagers, Harrison mentions that several biology teachers reported success using poppet bead models of genes and chromosomes for genetics studies. Once a popular children’s toy with which to make jewelry, poppet beads are small plastic balls, about 10 mm in diameter of varying colours (Figure 3). Each bead has two sides: a male connecting extension, and a female socket which together enable chains of poppet beads to be easily constructed into necklaces, bracelets and if needed, chromosomes. Poppet beads are readily purchased from biological product suppliers.

Any introduction to genetic algorithms should always pay passing tribute to the mechanism of natural selection, more commonly known as ‘survival of the fittest’. This can be easily achieved by using the poppet bead models from senior biology classes. One can start with the historically interesting story of *Biston betularia* moths. In the nineteenth century, butterfly and moth collecting was a popular pastime. In the 1850s, avid collectors would easily acquire specimens of the lightly coloured form of the moth known as *Biston betularia typica* but the more difficult to find sooty coloured variant *Biston betularia carbonaria* was even more highly prized (Figure 4). Some fifty years later the reverse was true. Examination of English museum butterfly collections from the turn of the twentieth century show that by then, the dark coloured moths were far more common. How could this have happened?

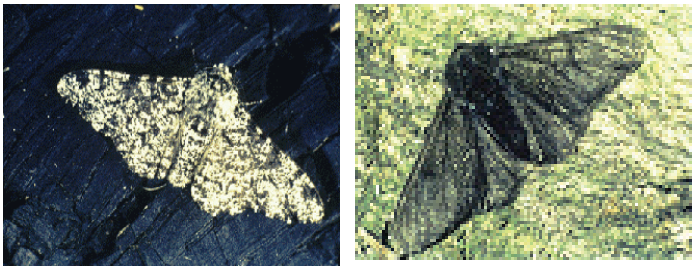

Figure 4: Two coloured variants of the same English moth: *Biston betularia typica* and *Biston betularia carbonaria*

Although coming under some recent academic scrutiny (Hooper, 2002; Wells, 1999), Kettlewell’s original experiments from the 1950s suggested that air pollution created by England’s industrial revolution was responsible for the phenomenon (Kettlewell, 1955). His widely quoted experiments supported the hypothesis that the darker moths were twice more likely to survive in the polluted woods than their lighter counterparts, as they avoided bird predation by being better camouflaged against the blackened tree trunks (Figure 5).

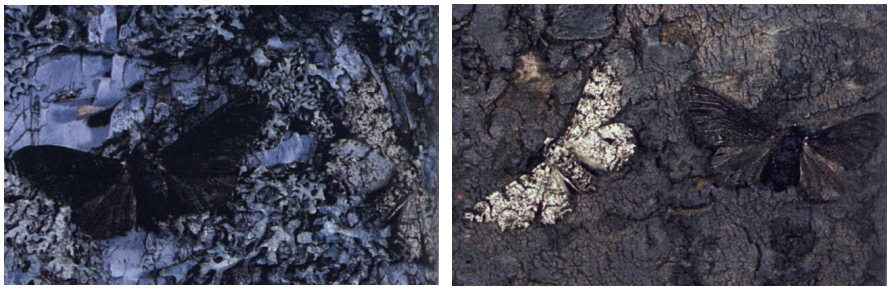

Figure 5: Two coloured moths on
a) a lichen covered tree trunk; b) a soot covered tree trunk

The First Exercise

Using poppet beads, it is easy to model a random population of moths. A moth can be easily represented by a string of black and white poppet beads. The proportion of white to black beads in any one string/moth is totally random but for reasons that will become obvious later, we suggest that each string consist of 8 beads.

Working in pairs, students are asked to commence with a population of eight moths, known in biology as the founding population, F_0. By examining each moth, students decide fitness in a population where being black is an advantage. By counting the number of black beads in the poppet bead string, a fitness score can be assigned to each moth. In Figure 6, the top left hand moth has a fitness of 3 and the bottom right moth has a fitness of 5.

Figure 6: Eight 'moths' of varying fitness in a founding population, F_0.

Students are asked to play 'god' and select the fittest 4 individuals and place them aside as members of the next generation, F_1. After discarding the remaining less fit moths, the students are instructed to use new beads to clone, that is make copies of the best 4 individuals. Once this has been done, the 4 clones are randomly divided into 2 pairs of 2, which will mate by the mechanism of 'crossover' (Figure 7). A crossover occurs when a random break is made in the string of poppet beads at the same location in the two parents. The beads from one parent are joined with the complement of beads from the other parent, and vice versa to create new offspring that are a ge-

netic mix of their component parents. The 4 resultant moths from these two matings are added to the F_1 population.

Figure 7: Two clones crossing over.

Finally, the last step in creating the F_1 generation of moths is done by randomly selecting one moth from the 8 and locating a single poppet bead in the string. A mutation is done on the poppet bead by changing the bead either from black to white, or white to black (Figure 8).

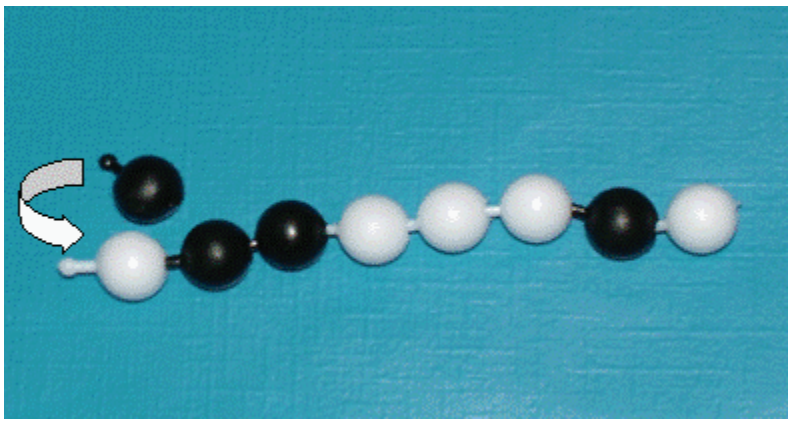

Figure 8: A mutation of one bead

It is possible to cover the key terminology for manipulation in a genetic algorithm by simply describing the above process to students as they create their moths for the F_1 generation. By iterating the procedure described above, the mechanisms of cloning, crossover and mutation are reinforced for students as they fashion the next generation, being the grandchildren of generation F_0 and labeled F_2. Meanwhile, the overall fitness of the population is visibly changing with the moths becoming collectively blacker (Figure 9). If the exercise is conducted in a tutorial or laboratory setting, a quick look around the room will illustrate to students the stochastic nature of the process, that is, no two groups started with the same F_0 population. Regardless, by applying 'survival of the fittest' selections, each generation converges towards a near black population.

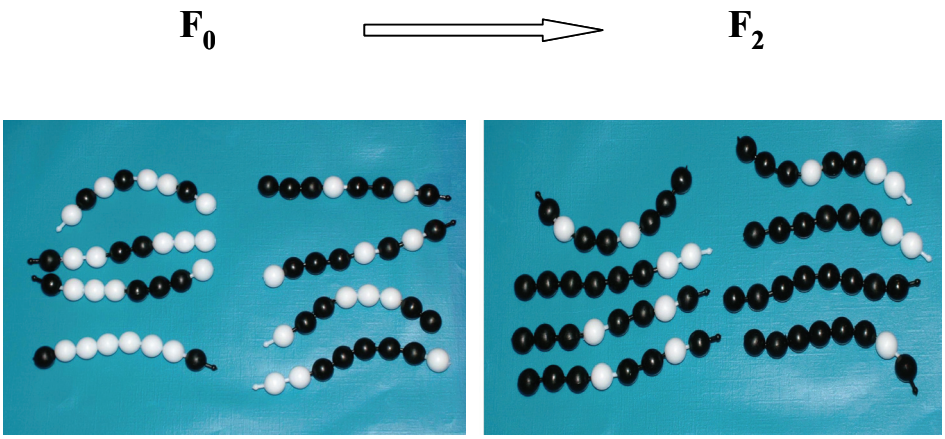

Figure 9: The commencing F_0 generation and after cloning, crossover and mutation the resulting F_2 of grandchildren

Students are asked to predict what the population will look like in the F_3 and F_{10} generations. Naturally, a discussion follows on the consequences for the moth population should conditions change and whiter moths are advantaged by selection. Additionally, at some stage the term chromosome should come into conversation as an alternate name for the moth or the string being manipulated.

The Follow-On Exercises

Next, students are introduced to a new problem, a computing problem rather than a biological one. The task is to find all the numbers between 0 and 255 that have a binary representation where the total number of 0s equals the total number of 1s. For example, 240 is one such number with a binary representation of 11110000, as is 153 with a binary code of 10011001 and, with some difficulty, it may be possible to iterate all the possibilities. By directing students towards a string representation of using a set of 8 bits to represent any number between 0 and 255, students may be reminded of the poppet bead model. In this new problem, white beads can be used to represent 0 and the black beads represent 1 and eight beads in a row is a simple representation of a number in the range 0 to 255.

As in the moth exercise, students working in pairs randomly make a population of 8 different numbers for an F_0 generation. Next a decision about the calculation of an appropriate fitness function needs to be made. In this task, it is not simply the number of black poppet beads in a string, but the ratio of black beads when compared to white beads, that matters for each individual number/chromosome being represented. A simple subtraction would suffice as a measure of how close the number of black beads to white beads is. If the number of black beads minus the number of white beads is zero, then the string represents a number that solves the problem. By repeating the above procedures of cloning, crossover and mutation on the randomly generated F_0 population, it is possible for students to quickly breed within a few generations, a population of chromosomes all with a low fitness value of 0. An important point for discussion with students is the question about when to stop producing new generations. In a typical GA, termination is usually decided when an end point has been reached such as all members of the population have the desired fitness, as in the above example. Alternatively, a set number of generations is decided, as in the moth example.

The above exercises illustrate several lessons to be learned by students. Firstly, when using a genetic algorithm there are only two design issues to consider: *What is a suitable chromosome*

representation for the problem? and *What is a suitable fitness function?* Secondly, GAs are totally stochastic processes where the mechanism of ‘survival of the fittest’ quickly ensures that randomly generated chromosomes evolve towards optimal solutions. Thirdly, genetic algorithms will perform equally well whether a maximum or a minimum is desired in the problem.

A third problem, that of solving mathematical equations using genetic algorithms is then introduced to students. Using a relatively simple equation, for example

$$z = x^2 - 2y + 3$$

students are asked to find the minimum value that z may take if x and y can only have integer values between 0 and 7? A discussion of possible strategies to solve this equation may suggest a trial and error approach or partial differentiation as methods.

In using a genetic algorithm for the above example, students are reminded of the two main design decisions, that is finding a suitable chromosome representation and an appropriate fitness function. In this instance, the equation for z is a suitable fitness function in deciding ‘survival of the fittest’. For the chromosome representation, it is possible to represent both x and y side by side on the chromosome as different genes. Since x can only have 8 possible values, from 0 to 7 inclusive, a 3 bit representation would suffice for all the possibilities. For example, if x = 0 this would be represented by 000 and if x = 7 by 111. Likewise, the values for y would also require 3 bits. Hence, creating a suitable chromosome for this problem requires a 3 bit + 3 bit = 6 bit string representation. Figure 10 shows a randomly generated chromosome representation of the values of x = 5 and y = 2. After randomly generating a small number of individuals then a poppet bead or paper exercise over a few generations will find the minimum solution quickly, being a minimum value for z at –11 when x = 0 and y = 7.

Figure 10: An example of a 6 bit chromosome representation when x = 5 and y = 2

An important concept in genetic algorithms is that of schema. A schema is the pattern of 1s and 0s on a chromosome that is manipulated by a genetic algorithm. In the classroom it is possible to ask students for their best solution and get different answers for z from different groups. Typically, fitness values of -11, -10, -9 and –8 may be reported. Examination of these ‘good’ fitness values and the chromosomes that yield them is shown in Figure 11. Attention can be drawn to the pattern of bits that these ‘good’ solutions have in common. For this exercise, the first two bits must be 0, and the fourth and fifth bits must be 1. This pattern is known as the schema for this particular problem.

A discussion about the concept captured by schema representation is valuable. Good schemas are affected by the genetic operators of crossover and mutation in the manipulation of a GA. Crossovers tend to preserve good schemas and mutations tend to disrupt them. When students move onto implementing and running GAs on their computers, decisions will need to be made as to an appropriate percentage of mutation and crossovers for each generation. Typically, most GAs commence at about 70% for crossover rate but a much smaller rate, like 1% for mutation. It is important that students know that they are expected to adjust these amounts during various trial runs of their software.

Figure 11: Best four solutions to the problem in $z = x^2 - 2y + 3$

Further Exercises

From the above introductory exercises it is possible to advance in different directions, dependent on the intention for introducing students to genetic algorithms. We teach GAs as a small unit within an artificial intelligence course, where they are compared to other problem solving strategies such as expert systems, fuzzy logic and neural networks. Another valuable resource for teaching genetic algorithms is the Internet where it is a simple matter to find applets that visibly reinforce the main GA terminology and mechanism covered above. These applets are wonderful tools to illustrate a wider range of genetic operations and evolutionary approaches to students. One such example is http://www.rennard.org/alife/english/gavgb.html.

Our next step is to supply students with three small Java classes, one to run a genetic algorithm, another to represent individual chromosomes and another one to handle report writing. Students run these classes in solving one of the standard GA test problems known as the Sphere Model where

$$f(x) = \sum_{i=1}^{N} (x_i - 1)^2$$

Whilst running the test problem, students experiment with the genetic algorithm by varying the number of generations, the number of individuals in each generation, the clone proportion, mutation rate and the crossover rate. Examination of the code illustrates to students that the construction of the coded genetic algorithm closely follows their classroom experiments; there are genes on chromosomes and a fitness function to do the 'survival of the fittest' selections. From these laboratory experiments, it is possible for students to explore more difficult problems such as timetabling and scheduling problems (Negnevitsky, 2005, p. 235).

In our artificial intelligence course, we always set a major open-ended assessment where students have free range to try and solve a problem using any of the approaches covered in classes. Last year, the problem was to research and code an intelligent system that could solve Sudoku problems (Sudoku, 2005; Venables & Tan, 2006). This year, students were told to investigate and

write software that could compose music. In both years, many students have chosen to apply a genetic algorithm in their assignments, with varying results.

About the Sudoku problem, one student reported that "*You don't need to be a genius to figure out that a GA stinks for solving this puzzle"* adding that "*Genetic Algorithms (GAs) cannot effectively solve problems in which there is no way to judge the fitness of an answer other than right/wrong, as there is no way to converge on the solution"*. The experiences of the students attempting the Sudoku problem were instructive in lectures this year, for underlining that GAs are useful in optimisation problems, but not always suitable in other circumstances. However, in their music composition efforts, GAs were quite useful to many students as a mechanism to evolve more melodic music from randomly generated notes. In an assignment submission, one student wrote *"In my code, I repeated the simple cloning process we used in class last week with the pop-it beads as my chromosome breeding mechanism."*

The Constructivist Approach

After introducing genetic algorithms for the first time with the above exercises, the teaching staff remained curious about the impact of this new approach. So to try and capture a snapshot of student perceptions of the initial poppet bead exercise, a simple survey instrument was designed. At the end of the unit, some 7 weeks after the introductory exercises were run, the students were asked to return an anonymous survey in which the following questions were asked:

- *Was the poppet bead exercise helpful to your understanding of genetic algorithms?*
- *Was the exercise unhelpful?*
- *Did you use a genetic algorithm in your assignment?*
- *Do you have any other comments about the poppet bead exercise?*

Twenty surveys were returned and with one exception, they were uniformly enthusiastic about the approach. Overall, the first question tended to be answered in most detail. Some typical responses were *"It was taught in a way which didn't make it seem difficult which was excellent"*, *"Hands on experience is always better than just theory"* and "*Yes, I felt it helped me visualise exactly what the lecturer was explaining"*. There seemed to be little correlation between the degree of enthusiasm shown in response to first question and the uptake of genetic algorithms as a strategy for their assignment queried in the third question. However the responses to the final question were more telling. Several students mentioned that they had used the methods learnt in the poppet bead exercise as the basis for their coding in the assignment.

Such active and contextual learning as indicated by students, is one of the hallmark characteristics of the constructivist model of learning widely reported in the educational literature. When students engage in problem-based and experiential learning they become active participants in their own learning. By arousing cognitive puzzlement, in this instance via their interactions and manipulation of poppet beads, students are stimulated in the synthesis of their own internal knowledge (Ben-Ari, 1998; Greening, 2000; Kolb, 1984; Kolb & Fry, 1975; Neo & Neo, 2001; Savery & Duffy, 2006; Tam, 2000).

In the science and mathematics education literature, constructivism is a powerful explanation of understanding student learning, and it is applicable also to computing education (Ben-Ari, 1998; Harrison, 2001). More specifically, Greening (2000) adopted an "ill-defined" large programming task to create an environment conducive to student learning through the practical dimensions of constructivism. Like Greening (2000, p. 95), we aim to achieve positive learning outcomes using the poppet bead exercises through the following:

Cognitive puzzlement – For information technology students the use of a 'hands on' analogical model is an unusual and unexpected experience. The quirkiness of using a once popular children's toy to understand the language from genetics and its application in genetic algorithms is a very useful stimulus to generate interest about GAs.

Inter-disciplinary problem domain – The cross fertilization of differing scientific disciplines is particularly pertinent in the study of GAs. As mentioned earlier, commencing students need an appreciation of both the biological and computing meanings of words, like chromosome and generation, to be comfortable in designing and using genetic algorithms as a problem solving strategy. Along the way, students develop a broader appreciation of some historical aspects of evolutionary biology and computing science and their relevance to each other.

Group work friendly – The introductory exercises are best carried out in pairs to enable easy manipulation of the bead model. By organizing and encouraging each other whilst trying to manipulate, crossover, clone and mutate their beads/moths/chromosomes students actively discuss the terminology and various aspects of the processes being simulated. Additionally, these exercises are followed by an open-ended, and perhaps "ill-defined", programming assessment task where students are required to complete as a pair.

Authenticity – The size and nature of the introductory exercises does not detract from the legitimacy of the biological problem involving *Biston betularia* moths nor the computing and mathematical problems that follow. In all cases, they are classical problems worthy of attention.

Ownership – Surprisingly, students do verbally express ownership regarding their moth populations as they go about creating and destroying individuals. Such ownership seems to come about due to their role as omnipresent beings that decide which chromosome strings are chosen for deletion, crossover and mutation. Less frivolously and more importantly students take ownership of the problem and their own learning when investigating and coding for their follow up assessments. As mentioned earlier, students are expected to assume responsibility for their chosen approaches and to experiment with the differing genetic algorithms and other strategies.

Stimulate reflection and meta cognition – As mentioned by Greening (2000), "*a lot of learning happens as retrospective".* We know this to be true from evidence we have gained from the research diaries that we ask students to record as they complete their open-ended assessments (Venables & Tan, 2006).

Encourage cognitive scaffolding – Using poppet beads to introduce genetic algorithms provided a framework in which students were able to discuss the concepts and mechanisms, safe in the knowledge that others had experienced the same situations. The framework was versatile in that it could be used to express biological concepts and be applied to computing problems. Many students in their assignment submissions made reference to the introductory exercises as the basis for their decisions regarding coding and experimentation.

Conclusion

"An algorithm must be seen to be believed."

Donald Knuth

Although Knuth was not referring to our approach for introducing genetic algorithms to undergraduate students, in the light of constructivist model of student learning, his quote is nonetheless pertinent. The described set of 'hands on' exercises allow students to build their own internal knowledge using problems that span biology, computing and mathematics. Together they illustrate the cross disciplinary nature of GAs and they introduce the associated 'biological' language and the underpinning concepts in a natural and meaningful way.

For undergraduates, genetic algorithms are an important heuristic strategy to teach. GAs naturally give a contextual position of computing with other disciplines. GAs can be used to expand discussion on topics like data structures, hill climbing searches and path finding algorithms. As well, by sparking student interest, it is hoped that these exercises will act as a springboard for exploring more about evolutionary algorithms and possibly other biologically inspired approaches such as swarm intelligences and anthill search algorithms.

References

Ben-Ari, M. (1998). Constructivism in computer science education. *Proceedings of the 29th SIGSCE Technical Symposium on Computer Science Education (SIGSCE 98)*, *30*, 257-261.

Greening, T. (2000). Students seen flocking in programming assignments. *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education,* Helsinki, Finland, 93-96.

Harrison, A.G. (2001). Thinking and working scientifically: The role of analogical and mental models. *Proceedings of the Australian Association for Research in Education Conference*, Fremantle, W.A., 2-6 December 2001. Retrieved 24 August 2006, from http://www.aare.edu.au/01pap/har01126.htm

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor.

Hooper, J. (2002, May 11). Of moths and men: Intrigue, tragedy and the peppered moth. *The Guardian Newspaper Review*.

Kay, A. (2006). Is computer science an oxymoron? Retrieved 27 October 2006 from http://www.windley.com/archives/2006/02/alan_kay_is_com.shtml

Kettlewell, H.B.D. (1955). Selection experiments on industrial melanism in the Lepidoptera. *Heredity*, *9*, 323-342.

Kolb, D.A. (1984). *Experiential learning: Experience as the source of learning and development*. Englewood Cliffs, NJ: Prentice Hall.

Kolb, D.A. & Fry, R. (1975). Towards an applied theory of experiential learning. In C.L. Cooper (Ed.), *Theories of group process.* Chichester: John Wiley.

Moore, M. (2001). Teaching students to use genetic algorithms to solve optimization problems. *Journal of Computing in Small Colleges. 16*(3),19-25.

Neo, K.T.K & Neo, M (2001). A constructivist learning experience: Reconstructing a web site using web based multimedia authoring tools. *Australian Journal of Educational Technology, 17*(3)*,* 330-350.

Negnevitsky, M. (2005). *Artificial intelligence: A guide to intelligent systems* (2nd ed.). Addison Wesley.

Savery, J. R., & Duffy, T.M. (2006). Problem based learning: An instructional model and its constructivist framework. Retrieved 10 October 2006 from http://www3.uakron.edu/edfound/people/savery/papers/sav-duff.html

Softpanorama. (2006). Algorithms and data structures. Retrieved 27 October 2006 from http://www.softpanorama.org/Algorithms/algorithms.shtml

Sudoku. (2005). Sudoku web site. http://www.sudoku.com/

Tam, M (2000): Constructivism, instructional design, and technology: Implications for transforming distance learning. *Educational Technology and Society, 3*(2), 50-60.

Venables, A. & Tan, G. (2006) Thinking and behaving scientifically in computer science: When failure is an option! *Journal of Information Technology Education, 5,* 121-13. Available at http://jite.org/documents/Vol5/v5p121-131Veneables138.pdf

Wells, J. (1999) Second thoughts about peppered moths: This classical story of evolution by natural selection needs revising. *The Scientist, 13*(11), 13.

Biographies

Anne Venables lectures in Computer Science at Victoria University, Melbourne, Australia. She has research and teaching interests in artificial intelligence and intelligence systems. Originally trained in Genetics, Anne spent several years as a secondary Science and Mathematics teacher before migrating into tertiary education. Anne is interested in innovations in education and has previously published in this field.

Grace Tan is a senior lecturer in Computer Science at Victoria University, Melbourne, Australia. Her research interests include investigations of innovative teaching methods, the development of graduate attributes, and issues related to female students in computing courses and Grace has published in these areas.