METHODOLOGY, PEDAGOGY, AND PHILOSOPHY

Erick Cantú-Paz, chair

On the Use of Negative Selection in an Artificial Immune System

Marc Ebner, Hans-Georg Breunig and Jürgen Albert
Universität Würzburg, Lehrstuhl für Informatik II,
Am Hubland, 97074 Würzburg, Germany
ebner@informatik.uni-wuerzburg.de, Tel. (+49)931/888-6612
http://www2.informatik.uni-wuerzburg.de/staff/ebner/welcome.html

Abstract

The natural immune system is very effective at protecting the body from diseases. Several researchers have analyzed the natural system and created artificial systems which copy mechanisms of the natural system in order to improve computer security. We suggest that the negative selection algorithm, which is at work in the natural system, might have been copied too closely. We argue against the use of negative selection if space is finite and self comprises only a small fraction of the available space or if space is infinite. We illustrate this on the problem of user authentication using keystroke analysis.

1 MOTIVATION

The natural immune systems' task is to detect molecules which don't belong to the organism. This ability led several researchers to look closely at the workings of the natural immune system. Inspired by the natural system they have tried to copy mechanisms which are at work in the natural system more or less closely for use in the area of computer security (D'haeseleer et al. 1996; Forrest et al. 1997; Forrest et al. 1996; Forrest et al. 1994; Hofmeyr and Forrest 1999a; Hofmeyr and Forrest 1999b; Kephart 1994; Kim and Bentley 2001a; Kim and Bentley 2001b; Somayaji et al. 1998). After having a closer look on how the natural immune system works, we briefly review some of the artificial immune systems and analyze the advantages and disadvantages of using the mechanism of negative selection in an artificial immune system.

2 THE NATURAL IMMUNE SYSTEM

Our discussion of the natural immune system is based on Alberts et al. (1994). A substance causing an immune reaction is called an antigen. The immune system is capable of distinguishing between highly similar antigens. Even proteins which differ by only a single amino acid can be distinguished. The cells which are responsible for the immune specificity are called lymphocytes. They belong to the class of white blood cells. The human body has approximately $2 \cdot 10^{12}$ lymphocytes. Two classes of lymphocytes exist: B-cells and T-cells. B-cells develop in the adult bone marrow or the fetal liver. They produce antibodies. T-cells develop in the thymus and are responsible for the so called cell-mediated immune response.

The immune system is based on a mechanism which is called clonal selection. Each lymphocyte is equipped with a receptor which can be used to bind an antigen. The term clonal selection comes form the fact that a large variety of receptors exist which can be grouped into families, or clones, of cell. Each receptor has a specific shape and can only react with a certain antigen. The receptors are generated at random and are thought to cover the whole space of possible antigens. If a lymphocyte binds an antigen, then the cell becomes activated. The cell proliferates, matures, and finally secretes antibodies. The antibodies have the same shape as the receptor of the cell which secreted it.

The antibody response includes the production of antibodies which circulate through the blood and other body fluids. The antibodies consist of a Y-shaped molecule which can bind an antigen at two locations. An abstract representation of this Y-shaped molecule and a close-up of the antigen-binding site of an antibody molecule is shown in Figure 1. The antibodies

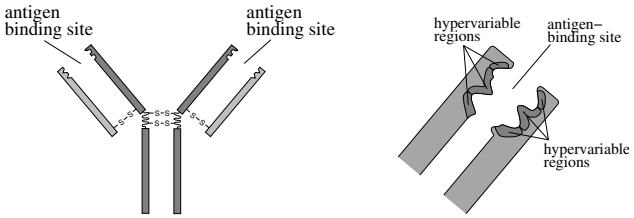


Figure 1: Antibody (left). Close-up of the antigen-binding site of an antibody molecule (right). Redrawn from Alberts et al. (1994).

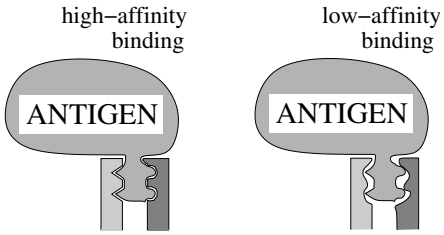


Figure 2: Binding of an antigen. Redrawn from Alberts et al. (1994).

bind antigens which fit into the receptors. Through this binding process a virus may be inactivated. Antigens coated with antibodies may also be digested or killed by special cells.

In the course of an immune response B-cells increase the affinity of the antibodies they produce. This process is called affinity maturation. Changes to the shape of the receptors are caused by mutations. This process is referred to as somatic hypermutation. The mutations happen with a frequency which is approximately one million times higher than the mutations which happen to the other genes. Cells which have a high affinity binding reproduce better because they can more easily dock on an antigen (Figure 2). This results in a selection of those cells which closely match the given antigen. Thus, an evolutionary process is embedded in the immune system which produces highly specific antibodies to any possible antigen.

The cell-mediated immune response consists of the production of specialized cells, called T-cells. These cells are used to detect cells which have been infected by a virus. Peptide fragments of a foreign molecule are brought to the cells surface by specialized molecules. Inside the cell those molecules are invisible to the immune system. Once these fragments show up on the cells surface they can be detected by the T-cells. We have two kinds of T-cells: cytotoxic T-cells and helper T-cells. Cytotoxic T-cells kill infected cell directly while helper T-cells activate other cells who then kill

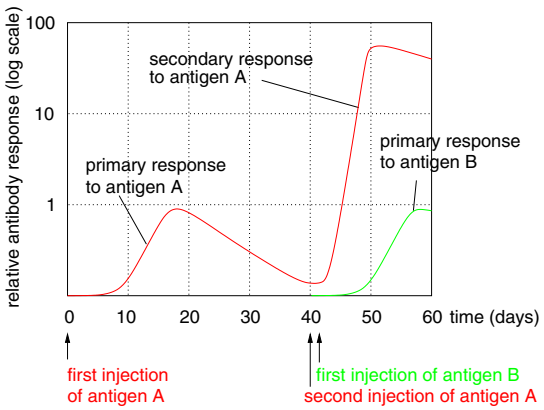


Figure 3: Response to antibodies. The response to the second exposure to antigen A happens much faster than the response to the first exposure. In addition the response is also stronger. Redrawn from Alberts et al. (1994).

the infected cells. In addition, the helper T-cells are necessary for the activation of B-cells. Helper T-cells stimulate themselves as well as other helper T-cells to reproduce once they are activated. Only those helper T-cells become activated which have detected an antigen.

In addition to the immune responses the immune system also has a memory. If an antigen is detected for the first time then the immune response only happens after a certain delay. This is called the primary immune response. The immune response on a known antigen, called the secondary immune response, happens quicker and more strongly in comparison to the primary immune response. The difference between the primary and the secondary immune response is shown in Figure 3. This behavior of the immune system is realized through different stages of the T- and B-cells. There are at least three different stages: virgin cells, activated cells and memory cells. Activated cells die after a few days. However, memory cells can live for several months or even years.

The main task of the natural immune system is to distinguish between own molecules and molecules belonging to a foreign organism. Detecting foreign molecules is mainly the task of the T-cells. The T-cells develop in the thymus. Cells which bind to own peptide are eliminated during development. This process is called negative selection. Only T-cells which have a low affinity to the organisms own molecules remain. B-cells need helper T-cells to react to foreign antigen. Therefore, helper T-cells also ensure that self-active B-cells are harmless.

Following this discussion of the human immune system we now have a look at how the workings of the natural system have been mapped to an artificial immune system.

3 PROPOSALS FOR AN ARTIFICIAL IMMUNE SYSTEM

In the area of computer security one needs to distinguish original data from manipulated data, authorized users from intruders and normal behavior from abnormal behavior. This is exactly the problem the natural system solves, namely to detect self from non-self. A number of properties of the natural system would also be useful for an artificial system: (a) distributed detection, (b) multi-layered, (c) diversity, i.e. every individual has its own immune system, (d) disposability, no single component is essential (e) the immune system can work autonomously, (f) is adaptive and (g) does not depend on secrets (Somayaji et al. 1998).

Kephart (1994) developed a biologically inspired immune system to protect a computer system from previously unencountered viruses or worms. The analogies between the natural system and the artificial system are rather loose. Integrity monitors in conjunction with activity monitors are used to determine if a virus or worm has entered the system. The integrity monitors check if files have been changed or added. Activity monitors check for dynamic behavior which is typical of viruses. They also look at the type of change to see if the change may have been caused by a virus. If it is determined that a virus has entered the system a scan is made to find any known viruses. In case the virus is known, it is removed. Otherwise, decoy programs are used in order to get a sample of the virus. This has been likened to the ingestion of antigen by macrophages or B cells in the natural immune system.

Forrest et al. (1994) developed an artificial immune system for change detection in executables. The system learns to distinguish the original version of a program from a program which has possibly been infected by a virus. Forrest et al. generate a set of random detectors (bit strings) in analogy to the workings of the thymus of the natural immune system. The negative selection algorithm is used to remove those detectors which would detect the original programs. The feasibility of generating detectors was analyzed by D'haeseleer et al. (1996) who also proposed a more efficient algorithm for generating detectors.

Hofmeyr and Forrest (1999a, 1999b) developed an artificial immune system for intrusion detection. This system has a closer analogy to the workings of the natural immune system. The system's task is to distinguish normal from abnormal connections between two computers in a local area network. The system basically consists of a set of detectors which are used to detect non-self, i.e. abnormal behavior. Initially the detectors are generated at random. During an initial maturation period, it is checked if a detector matches any part of the system which is to be protected. If a match occurs then the detector is deleted. If a detector has survived this process for a specified number of steps then the detector matures and is now ready to detect non-self. The set of mature detectors continually monitor the data stream for non-self. If a detector is not activated for some time then the detector is deleted and replaced with a new mature detector leaving the negative selection algorithm. Detectors are memorized if a detector receives a special type of co-stimulation. Detectors which have been memorized previously can immediately detect non-self.

Forrest et al. (1996) have developed an artificial immune system which monitors dynamic behavior of processes. Sequences of system calls are used to define normal behavior for standard unix programs. Deviations from this normal behavior are detected by comparing short sequences of system calls with normal sequences stored in a database. Plans to extend this work include the addition of the negative selection algorithm and using on-line learning.

Kim and Bentley (2001b) modeled clonal selection for use in an artificial immune system for network intrusion detection. Basically, Kim and Bentley evolve detectors which detect non-self antigens. A negative selection operator is embedded in this process. Detectors which match self antigens are deleted. Other authors have used principles from artificial immune systems for diversity maintenance in multi objective optimization (Cui et al. 2001), to improve adaptability in the context of time dependent optimization (Gaspar and Collard 1999) or to recognize spectra for chemical analysis (Dasgupta et al. 1999).

Many of the artificial systems stay very close to the workings of the natural system, with all its advantages and disadvantages. We now have a closer look at the efficiency of the negative selection algorithm.

4 ON THE EFFICIENCY OF THE NEGATIVE SELECTION ALGORITHM

Recently, Kim and Bentley (2001a) analyzed negative selection in an artificial immune system for intrusion detection. Their main result is that as the task be-

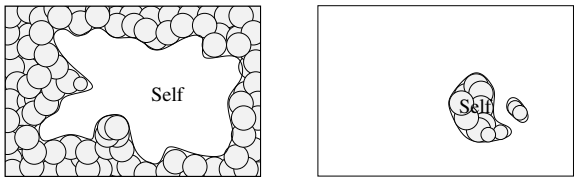


Figure 4: (a) Covering non-self with detectors works best if self is large and non-self occupies only a small fraction of space. (b) On the other hand, covering self with detectors works best if self is small and non-self occupies a large fraction of total space.

comes more complex, the number of detectors has to be unacceptably large and the time needed to generate a sufficient number of detectors is impractical. In comparison to this critique of the negative selection algorithm our critique is much more general.

For this analysis we assume that we have n-dimensional real valued vectors which represent antigens and detectors. We also assume, that we have some mechanism which is able to detect if a match occurred between a detector and an antigen. This can be some arbitrary measure such as correlation coefficient or distance between the two vectors. Now we need a definition of self. We define self as a subset of points in the n-dimensional space. The set of points describing self does not have to be static but can vary over time. A threshold is usually used to determine if a detector matches either an antigen or any point of the self. This fact is modeled by placing a hyper-sphere around each point which belong to the self. The radius of the hyper-sphere determines the sensitivity of the detector. An antigen is detected if it lies inside the hyper-sphere of a detector.

The negative selection algorithm distributes detectors randomly over this space. Detectors overlapping any points of self are removed (Figure 4a). This algorithm works fine if space is finite and self occupies a large fraction of the total space. But what if space is finite and self comprises only a small fraction of the available space or what if space is infinite? In this case it makes more sense to describe only the self and then check if an antigen falls outside of this area (Figure 4b).

Note that if the number of detectors is finite then learning a concept or learning its negation are not equivalent. If self can be covered using a smaller number of detectors in comparison to non-self then it makes more sense to detect self instead of non-self. If space is infinite and one tries to cover all points belonging to non-self, only a finite number of points will

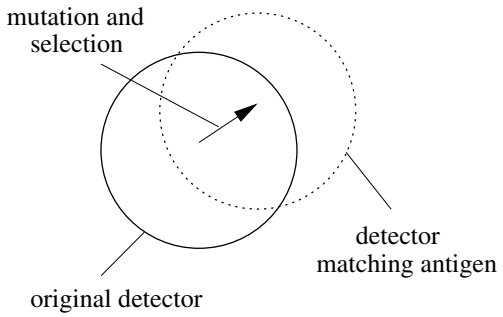


Figure 5: Somatic hypermutation and clonal selection increases the affinity between detector and antigen. This corresponds to a movement in shape space.

be covered. The sensitivity of the detectors need to be reduced if one wants to be able to detect all possible antigens. Reducing the sensitivity means enlarging the radius of the hyper-spheres. However this also means that during random generation of a detector it is very likely that this detector will cover some part of self and therefore will be removed from the set of detectors. If the size of self is small and the number of detectors is limited then self can be much better approximated by placing the detectors inside of self.

Suppose we want to determine if a point is located outside of a square. All we need is to check if the point is *not* located *inside* the square. This negation of a test can be done in a computer system quite easily but is very difficult to realize in a natural system. For the natural immune system it is simply not possibly to check if a given molecule is equivalent to one of its own molecules. The natural system would have to store samples of the molecules which occur in the human body in some part of the body and then check if the intruder falls *outside* this class of molecules. That is, the natural system would have to check that the given molecule is unlike any of the stored molecules. Because this is infeasible, the natural system is dependent on the use of negative selection to detect molecules which don't belong to its own organism. This works because local shape space is only finite (Kauffman 1993).

The natural system generates detectors which match other molecules rather crudely. The match is then refined using somatic hypermutation. Detectors which have a better match produce more offspring. What analog would there be in an artificial immune system for the process of somatic hypermutation? In our model, this would correspond to moving the detector in the direction of the position of the antigen (Figure 5). But is this really necessary? If it is established that some antigen is present in the system then it would be

character	duration	delay	character	duration	delay	character

Figure 6: Molecule generated from the user's keystrokes. Each molecule is composed of 7 fields.

the best to store the position of the antigen as a sample of an unknown intruder. The more samples are gained the better our knowledge of the intruder becomes.

So far, our discussion was rather general with no particular problem in mind. We now have a look at solving the problem of user authentication with an artificial immune system.

5 A PRACTICAL EXAMPLE

Biometrics such as a fingerprint or iris pattern may be used to determine who is actually using a computer system. For instance, Klosterman and Ganger (2000) have developed a system for continuous user authentication using a face recognition system. Other examples for user authentication include the analysis of keystroke patterns (Bleha et al. 1990; Brown and Rogers 1993; Furnell et al. 1996; Furnell et al. 1995; Joyce and Gupta 1990; Leggett and Williams 1988; Monrose et al. 1999; Monrose and Rubin 1997; Obaidat and Sadoun 1997; Robinson et al. 1998; Shepherd 1995; Song et al. 1997; Umphress and Williams 1985). As a practical example, we have chosen to analyze typing patterns. We want to make sure that the same user is continually sitting in front of the keyboard. For instance, if a person goes to lunch and forgets to lock the screen, then somebody not authorized could use the terminal. Keystroke analysis could also be used to make sure that even if the password is known to an intruder it can only be used to gain access to the system if the speed of typing corresponds to the authorized user. That is, we want to develop an artificial immune system for user authentication.

The system tries to determine if the same or a different user is using the system. This information is derived by analyzing the user's keystrokes. The duration of the key presses and the delay between key presses is used to determine who is typing on the keyboard. A stream of molecules is generated from the timestamps which are recorded whenever a key is either pressed or released. Each molecule contains data from three successive key release events. The data is stored in seven fields as shown in Figure 6. The first field contains the first character which was released. The second field contains the duration of the key press. The third field contains the delay between the release time and the time of the next key press. The fourth field contains the second character. The fifth field contains the duration of the second key press. The sixth field contains the delay between the release time and the time of the third key press and finally the last field contains the third character pressed.

First we need a notion of self. Self is defined as the normal typing behavior of a user. That is, all points in our 7 dimensional space which describe the typing behavior of the user belong to the set of self. All others belong to the set of non-self. In order to detect a different user we could randomly generate detectors and then check if the detectors match any of the molecules. If a detector matches any part of self then we delete this detector. The problem with this approach is how can we cover an infinite space? To solve this problem we have chosen to store samples of the normal typing behavior of the user and to check the stream of molecules against this sample. Thus, we do not use the negative selection algorithm.

For our experiments we have used a pool of 2000 detectors. Each molecule is stored in the pool of detectors after a delay of 2000 iterations of our algorithm. An activity level models the clonal reproduction of the artificial immune system. The stream of molecules is checked against this pool of detectors. If a match is made between a molecule and a detector, i.e. the molecule is sufficiently similar to one of the detectors, we decrease the activity level by 1% otherwise we increase the activity level by 1%. The activity level is set to 1.0 at the start of our algorithm. The activity can reach a maximum of 2.0. If the activity level reaches 1.5 (half-way between maximum and minimum values) then we assume that a different user has gained unauthorized access to the keyboard.

For a match between a molecule and a detector to be made we require that fields one, four and seven are equivalent. If these three fields are equivalent, then we look at fields two, three, five and six to determine how similar these fields are. For each of these four fields we calculate the following similarity measure

$$\text{similarity} = \sum_{i \in \{2,3,5,6\}} e^{-\frac{(a_i - d_i)^2}{\sigma_i^2}}$$

where a_i and d_i refer to the i-th element of the molecule (a possible antigen) respectively detector and $\sigma_i = d_i/2.1$. A match is made if the similarity measure reaches a value of 3.2 or higher.

We obtained typing characteristics for 5 different users. Each user had to type the first two sections of the seminal paper "Computing machinery and intelligence" by

Table 2: Number of keystrokes until change is detected.

User	1	2	3	4	5
1	-	141	100	139	267
2	39	-	67	459	46
3	53	52	-	71	54
4	123	322	209	-	113
5	120	101	131	119	-

Turing (1950). The data obtained from the keystroke timestamps was saved and converted into a stream of molecules. The stream of molecules was then analyzed offline. Each stream consisted of between 5168 and 5741 molecules. For each user we randomly selected a reading position from which we start reading molecules. After 2000 molecules we start reading molecules from the second users stream. Each user's stream of molecules was compared against the streams of all other users resulting in a 5×5 matrix of activity graphs. If a user's stream was compared against its own stream then we simply skipped 100 molecules after 2000 molecules have been processed. The results of all experiments are shown in Table 1. Table 2 lists the number of molecules (or keystrokes) until non-self was detected.

6 DISCUSSION

The results show that non-self is detected after a relatively small number of keystrokes. Simply storing samples of the user's typing behavior works well for the task of user authentication. However, this is not the only way to address this problem. Another possibility would be to average data and thereby to establish a model of the person sitting in front of the keyboard. In this case, the task is to obtain a compressed form of the data. For the task described here, this can be achieved by calculating the mean and the variance of the duration of keystrokes and the delay between keystrokes.

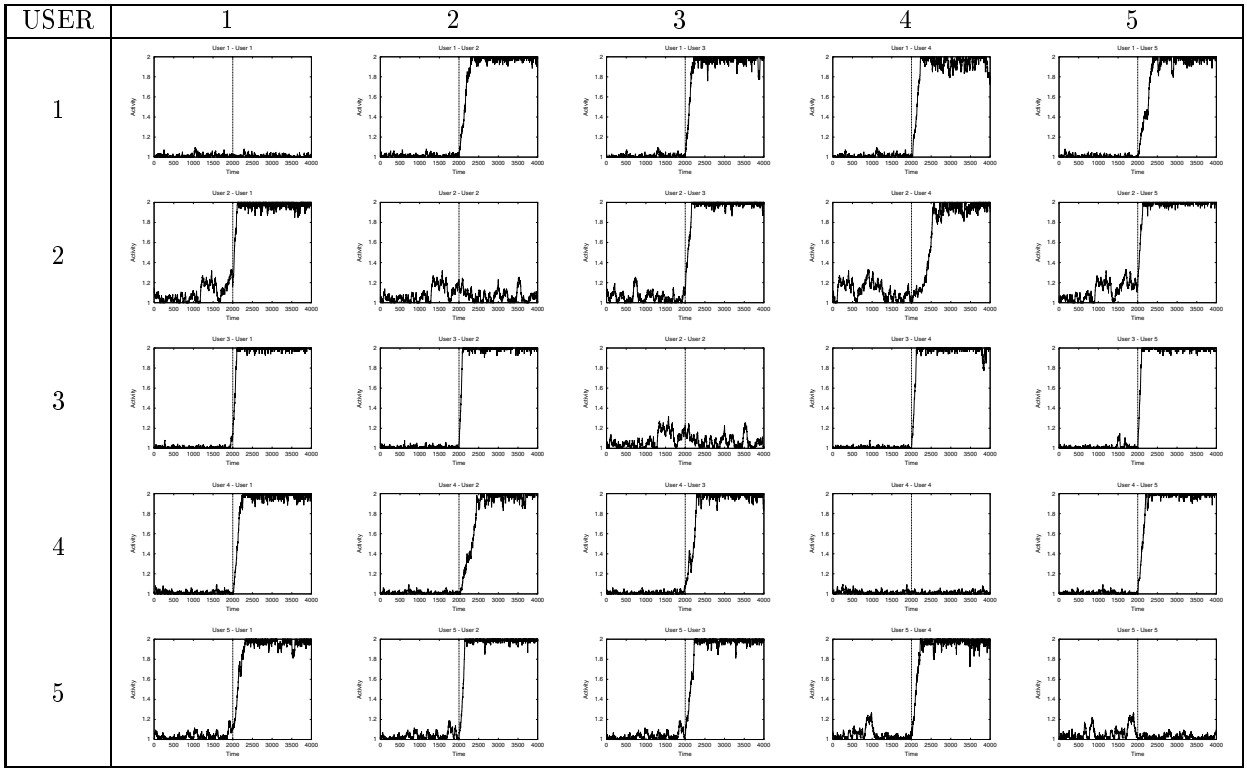
In fact, deriving a model from keystroke characteristics for user authentication was proposed by several researchers (Bleha et al. 1990; Brown and Rogers 1993; Furnell et al. 1996; Furnell et al. 1995; Joyce and Gupta 1990; Leggett and Williams 1988; Monrose and Rubin 1997; Obaidat and Sadoun 1997; Robinson et al. 1998; Shepherd 1995; Song et al. 1997; Umphress and Williams 1985). Song et al. (1997) use the same representation as we do for continuously monitoring the user's keystrokes. In particular, the mean and standard deviation of the duration of a key press and the delay between a key release and another key press are calculated over two or more consecutive press and release events. Next, the probability that the current duration of key presses and the delay between keystrokes belong to the current user is calculated. If the sum of probabilities is very low for a number of time steps then a different user is likely to be sitting in front of the terminal.

The main problem with model based approaches is how to treat outliers. Some researchers e.g. Joyce and Gupta (1990), Leggett and Williams (1988), and Umphress and Williams (1985) remove outliers. However the difficult task is to define what is an outlier. With our approach all user patters are stored. The authorized user will occasionally produce outliers but the large majority of typing patters will be consistent, i.e. not raise the activation value. In contrast, an unauthorized user will almost always produce outliers which keep raising the activation value of the system. Adaptation is achieved by storing molecules in the pool of detectors after a delay of several iterations. Thus our pool of detectors represents an undistorted view of the user's typing pattern. Furnell et al. (1995) and Furnell et al. (1996) achieve a detection rate of 85% within 160 keystrokes or less with a system based on statistical methods. Our system is simpler yet we detect 80% of the intrusion attempts within 160 keystrokes or less.

7 CONCLUSIONS

The natural immune system does a very good job of protecting the body from diseases. Analysis of the natural system can provide many paths to increased computer security. Several successful systems have already been proposed. In our research we focused on the role of negative selection in the immune system. The negative selection operator does a beautiful job in the natural system but is not necessarily useful in an artificial system. The natural system basically has no other way to detect foreign antigens than to remove those cells which produce antibodies which detect its own molecules. However, in a computer system we are able to determine easily if an element is *not* a member of a given set. Thus we don't need to invoke the negative selection algorithm here. As a practical problem to illustrate this case we have chosen user authentication using keystroke analysis. Samples of the user's typing characteristics were stored in a pool of detectors and compared with the current typing behavior. If the typing behavior deviates too much from the normal typing behavior then a different user is likely to be using the keyboard. Experimental results were provided for 5 different users. In each case non-self was

Table 1: Self and non-self detection for 5 users. Each graph shows the activity level over time. The graphs in the diagonal show how the system behaves for a single user when 100 molecules are skipped after 2000 molecules have been processed. Because the typing behavior is still the same, no change is detected. In contrast, the activity level rises sharply whenever a different user's stream is processed. The vertical line denotes the time when 2000 molecules have been processed.



quickly detected after a change to a different user occurred.

Acknowledgement

We thank Rob Shipman from BT Labs, UK, and Richard Watson from Brandeis University, USA, for helpful comments on the draft version of this paper.

References

B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson (1994). *Molecular Biology of the cell* (3rd ed.). New York: Garland Publishing.

S. Bleha, C. Slivinsky, and B. Hussien (1990). Computer-access security systems using keystroke dynamics. *IEEE Trans. on Pattern Analysis and Machine Intelligence 12*(12), 1217–1222.

M. Brown, and S. J. Rogers (1993). User identification via keystroke characteristics of typed names using neural networks. *Int. Journal of Man-Machine Studies 39*, 999–1014.

X. Cui, M. Li, and T. Fang (2001). Study of population diversity of multiobjective evolutionary algorithm based on immune and entropy principles. In *Proc. of the 2001 Congress on Evolutionary Computation, COEX, Seoul, Korea*, pp. 1316–1321. IEEE Press.

D. Dasgupta, Y. Cao, and C. Yang (1999). An immunogenetic approach to spectra recognition. In *Proc. of the 1999 Congress on Evolutionary Computation*, Mayflower Hotel, Washington D.C., USA, pp. 1859–1866. IEEE Press.

P. D'haeseleer, S. Forrest, and P. Helman (1996). An immunological approach to change detection: Algorithms, analysis and implications. In *Proc. of 1996 IEEE Symposium on Computer Security and Privacy*, pp. 110–119. IEEE Computer Society Press.

S. Forrest, S. A. Hofmeyr, and A. Somayaji (1997).

Computer immunology. *Communications of the ACM 40*(10), 88–96.

S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff (1996). A sense of self for unix processes. In *Proc. of 1996 IEEE Symposium on Computer Security and Privacy*, pp. 120–128. IEEE Computer Society Press.

S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri (1994). Self-nonself discrimination in a computer. In *Proc. of the 1994 IEEE Symposium on Research in Security and Privacy*. IEEE Computer Society Press.

S. M. Furnell, J. P. Morrissey, P. W. Sanders, and C. T. Stockel (1996). Applications of keystroke analysis for improved login security and continuous user authentication. In *Proc. of the 12th Int. Conf. on Information Security (IFIP SEC '96), Island of Samos, Greece*, pp. 283–294.

S. M. Furnell, P. W. Sanders, and C. T. Stockel (1995). The use of keystroke analysis for continuous user identity verification and supervision. In *Proc. of MEDIACOMM 95 - Int. Conf. on Multimedia Communications, Southampton, UK*, pp. 189–193.

A. Gaspar and P. Collard (1999). From gas to artificial immune systems: Improving adaptation in time dependent optimization. In *Proc. of the 1999 Congress on Evolutionary Computation*, Mayflower Hotel, Washington D.C., USA, pp. 1859–1866. IEEE Press.

S. A. Hofmeyrd and S. Forrest (1999a). Architecture for an artificial immune system. *Evolutionary Computation 7*(1), 45–68.

S. A. Hofmeyr and S. Forrest (1999b). Immunity by design: An artificial immune system. In *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 1289–1296. Morgan Kaufmann.

R. Joyce and G. Gupta (1990). Identity authentication based on keystroke latencies. *Communications of the ACM 33*(2), 168–176.

S. A. Kauffman (1993). *The Origins of Order. Self-Organization and Selection in Evolution*. Oxford: Oxford University Press.

J. O. Kephart (1994). A biologically inspired immune system for computers. In R. A. Brooks and P. Maes (Eds.), *Artificial Life IV: Proc. of the 4th Int. Workshop on the Synthesis and Simulation of Living Systems*, pp. 130–139. MIT Press.

J. Kim and P. J. Bentley (2001a). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proc. of the 2001 Congress on Evolutionary Computation, COEX, Seoul, Korea*, pp. 1244–1252. IEEE Press.

J. Kim and P. J. Bentley (2001b). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Genetic and Evolutionary Computation Conference 2001, San Francisco*, pp. 1330–1337.

A. J. Klosterman and G. R. Ganger (2000). Secure continuous biometric-enhanced authentication. Technical Report CMU-CS-00-134, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

J. Leggett and G. Williams (1988). Verifiying identity via keystroke characteristics. *Int. Journal of Man-Machine Studies 28*, 67–76.

F. Monrose, M. K. Reiter, and S. Wetzel (1999). Password hardening based on keystroke dynamics. In *6th ACM Conf. on Computer and Communications Security, Kent Ridge Digital Labs, Singapore*, pp. 73–82. ACM Press.

F. Monrose and A. Rubin (1997). Authentication via keystroke dynamics. In *4th ACM Conf. on Computer and Communications Security*, pp. 48–56.

M. S. Obaidat and B. Sadoun (1997). Verification of computer users using keystroke dynamics. *IEEE Trans. on Systems, Man, and Cybernetics – Part B: Cybernetics 27*(2), 261–269.

J. A. Robinson, V. M. Liang, J. A. M. Chambers, and C. L. MacKenzie (1998). Computer user verification using login string keystroke dynamics. *IEEE Trans. on Systems, Man, and Cybernetics – Part A: Systems and Humans 28*(2), 236–241.

S. J. Shepherd (1995). Continuous authentication by analysis of keyboard typing characteristics. In *European Convention on Security and Detection, 16-18 May*, pp. 111–114. IEE.

A. Somayaji, S. A. Hofmeyr, and S. Forrest (1998). Principles of a computer immune system. In *1997 New Security Paradigms Workshop*, pp. 75–82. ACM.

D. Song, P. Venable, and A. Perrig (1997). User recognition by keystroke latency pattern analysis.

A. M. Turing (1950). Computing machinery and intelligence. *Mind 59*, 433–560.

D. Umphress and G. Williams (1985). Indentity verification through keyboard characteristics. *Int. Journal of Man-Machine Studies 23*, 263–273.

Evolutionary Computation as a Form of Organization

Alexander Kosorukoff
Dept. of General Engineering
University of Illinois
Urbana-Champaign, IL 61801
alex3@illigal.ge.uiuc.edu

David Goldberg
Dept. of General Engineering
University of Illinois
Urbana-Champaign, IL 61801
deg@illigal.ge.uiuc.edu

Abstract

Traditional areas of application of genetic algorithms (GA) are engineering and technology. Success of genetic algorithms there is well known. This paper explores the use of genetic algorithms as models to influence the design of organization. In particular, we outline the concept of evolutionary organization process based on two recent cases: the Teamwork for a Quality Education (TQE) and Free Knowledge Exchange (FKE) projects. The distinguishing feature of both projects is that computational evolutionary processes influence the organizational environment, providing the structure of interactions of people and facilitating their communication. In both cases, the organizational structure and people become directly involved into the evolutionary process integrating the power of evolutionary computation with the competence of participating human beings.

1 INTRODUCTION

Traditionally, evolutionary computation (EC) is considered separately from the organizational environment in which it operates. The organizational environment provides the problem to be solved and the fitness criteria of solutions. As a result of EC execution a population of good-enough solutions is created that feeds back into the environment. In this mode of operation, EC does not influence the structure of organizational environment from which it is invoked. Instead, it serves just as a functional unit of a fixed structural mechanism. However, if we look beyond one run of the EC process, the organizational environment has people who make changes to the problem and the fitness function. This can be viewed as another process of evolution: the evolution of human ideas or memes (Dawkins, 1976). Intuitively, the two processes of evolution, computational and human, have the same nature. Several authors suggested that genetic algorithms model human innovation (Goldberg, 1983; Holland, 1995; Goldberg, 2000). If so, the efficient and convenient interface between the two might speed up the evolution of the whole human-computer system. This paper considers two applications which provide such an interface, creating a fusion of computational evolution and the evolution of human thoughts. Such hybrid evolutionary process is interesting as (1) a method of studying innovative behavior of humans, (2) a natural method of embedding the competence of human users into an evolutionary procedure, and (3) an organizational method to improve the innovation ability of a group of people.

In this paper, we consider two applications of GA principle for social organization. The first project "Teamwork for a Quality of Education" (TQE) is a method to organize an educational process after the model of a genetic algorithm. The second project is "Free Knowledge Exchange", the web-based virtual organization that uses a human-based genetic algorithm (HBGA) (Kosorukoff, 2001) for its internal knowledge management and innovation.

2 TEAMWORK FOR A QUALITY EDUCATION (TQE) PROJECT

The TQE project was an application of the basic concepts of a genetic algorithm to create a more efficient educational environment (Goldberg, Hall, Krussow, Lee, & Walker, 1998). It introduced teamwork and design across the curriculum enlivened by a spirited, yet friendly competition among teams. It also defined the principles, projects, and the rules of the competition.

TQE is a competition of student-led teams, each team consisting of freshmen, juniors, and seniors together with faculty and staff advisors. Among TQE principles, the following three are most important for our analysis:

- **Pervasive Teamwork** To achieve higher quality delivery of engineering education, integrated teams should be employed throughout the engineering academy as they have been employed in industry.
- **Friendly Competition Among a Population of Teams** Participation and excellence should be driven by a friendly competition among a population of teams to win team awards based on excellence in academics, projects, and other categories.
- **Multiobjective evaluation** Each team is charged with obtaining the highest quality education possible for its members, and this goal is actuated through the series of competitions in three broad categories: (1) academics, (2) service and design, and (3) summer job placement.

TQE provides participants with a structure of interaction built upon the principles of a GA. In the common academic approach, faculty, staff, and students are like billiard balls that collide with one another when a course, advising episode, or other event calls for it. Under TQE program, the same collisions would take place, but the individuals would also be supported by a quasi-permanent interpersonal infrastructure of teamwork (Goldberg, Hall, Krussow, Lee, & Walker, 1998).

Additional information can be found elsewhere (Goldberg, Hall, Krussow, Lee, & Walker, 1998), but for the purpose of this paper the key thing to keep in mind is that the TQE was conceived partially because of the second author's experience with GAs. In other words, the very notion of a population of teams, a competition, a fixed (and multiobjective) "fitness function" were drawn from the example of GAs. Additionally, it was assumed that teams would emulate one another, thereby promoting a kind of selection and crossover. It was also assumed that a team member would spontaneously generate new and useful ideas, a kind of smart mutation.

While the common academic approach is oriented toward the development of the individual abilities of a student, TQE emphasizes the development of cooperative skills. Individual grading, the fitness function of usual education, is augmented by team grading, so the educational process optimizes the performance of a team rather than the peak performance of an individual. This produces diverse teams capable of solving tasks the complexity of which is beyond abilities of an individual specialist.

Table 1: Correspondence between elements and procedures of GA and TQE

GA	TQE
Gene	Member
Chromosome	Team
Population	Population of teams
Fitness function	Judging + grades
Generation	Semester
Initialization	Team formation
Selection	Team competition
Crossover	Team swaps + informal exchange
Mutation	New idea of a team member

A detailed description of the results of the TQE pilot project is beyond the scope of this treatment. Student feedback was generally good though the course required a substantial workload for the credit given. Detailed description of the results with the program is available elsewhere (Goldberg, Hall, Krussow, Lee, & Walker, 1998). Here we concentrate on the GA-connection.

The TQE project was inspired by a genetic algorithm, and there is a strong correspondence between its concepts and the concepts of a GA. This is represented in table 1

Most of the table is self-explanatory. New idea creation by a team member is analogous to a mutation of one gene, which is a team member in this case. The correspondence between TQE and GA procedures is pretty strong, except for the following two differences: crossover and reproduction operators.

The usual kind of GA crossover is difficult to apply to a team of people, because team members unlike genes have their own preferences and desires. Therefore, we cannot just swap members randomly between two teams. The practice of team swaps is tightly connected with the willingness of particular members to change their team, and usually this process happens actively only at the initial stages of TQE. After the teams become more or less solid, crossover rarely happen, so its combinatorial potential cannot be fully utilized.

Fitness-proportional reproduction is another problem

in TQE. If some genotype is fit in GA we can easily reproduce and make several copies of its genes. It is clear that we cannot clone people of the fit team in the same way.

Concluding this section, we note that generally TQE is a working method of organization built after the GA model with some modifications reflecting social specifics of this project.

3 FREE KNOWLEDGE EXCHANGE (FKE) PROJECT

The Free Knowledge Exchange (FKE) project introduces the concept of evolutionary knowledge management based on concepts of GA. It used a human-based genetic algorithm (HBGA) for the task of collaborative solving of problems expressed in natural language (Kosorukoff, 2000a). It was created in 1997 for a small organization with the goal of promoting success of each member through new forms of cooperation based on better knowledge management. Currently it is a virtual internet community of more than 500 people from 92 countries. The FKE website www.3form.com evolves solutions to the problems of its participants in 7 different languages. It is supported by advertisement and allows anyone to join this community through the web and use it without a membership fee.

The FKE project explores evolution of natural language strings to arrive at better answers to the problems submitted by its members. It organizes individuals into collaborative community and uses their ability to perform intelligent crossover and selection operators on existing knowledge.

The idea of knowledge evolution in the most explicit form was suggested by Richard Dawkins (Dawkins, 1976). Evolution of natural language messages was explored in neuro-linguistic programming (Bandler & Grinder, 1976) and studied in the evolutionary theory of language (Pinker, 1998). Some web projects implicitly use evolution of messages to stimulate creativity, and the most relevant example is the Global Ideas Bank (GIB). Its main idea is collecting more successful and humane ways of doing things, and then re-presenting them in new mixes and matches, the accumulation of systems and arrangements that work a little better (Eno, 1998).

FKE makes the evolution of messages systematic and explicit using the framework of evolutionary computation. The four main ideas and their sources are shown in table 2: human interaction, emphasis on recombination, using natural language as a genotype, and diploidy.

Table 2: Main ideas behind FKE and their sources

Idea	**Source**
Human interaction	IGA
Natural language as a genotype representation	Meme theory by Dawkins
Emphasis on Recombination	GA
Diploidy	Creative questioning

The idea of human interaction came from interactive genetic algorithms (IGA) that introduced human evaluation interfaces in evolutionary computation. Human-based genetic algorithm (HBGA) used in FKE is basically an IGA combined with human-based innovation interfaces (crossover and mutation). In comparison with a typical interactive genetic algorithm using only human judgment (Herdy, 1996; Takagi, 1998), HBGA enjoys a balanced approach allowing and encouraging both convergent and divergent thinking of participants in the form of evaluation and recombination, correspondingly. This is accomplished by selecto-recombinative interfaces, where a person can perform selection or evaluation or both, based on one's preferences at the particular moment.

The use of natural language as a genotype representation was inspired by meme theory (Dawkins, 1976). It can be said that natural language strings have a tight linkage of building blocks, since in the most frequently used patterns of language their constituent parts tend to be located close to each other. Thus our natural language by itself has a good encoding for the purpose of genetic algorithm. We can hypothesize that the structure of language has itself evolved to allow this tight linkage of building blocks. Such a structure makes the evolutionary method of natural language processing efficient.

The emphasis on recombination is the main feature of GA. From an EC point of view, online interfaces of Global Ideas Bank are built on selection and mutation. They show ideas one at a time and allow users to evaluate them, to add new ideas and comments. Human-based mutation happens when one idea inspires another, and the result is submitted back to the ideas bank. This mechanism of evolution produces incremental continuous improvement, but lacks efficiency without recombination operator (Goldberg, 2000). The FKE project puts much emphasis on the process of recombination as do many GA implementations.

Table 3: An example of evolved answers in FKE

Q	How do you know that you are in the right career to really exceed in life and to do well overall?
18	You are in the right career if you wake up excited to go to work.
12	You need to feel it. If you're enjoying yourself, nothing else matters. As long as the money is enough to live by, you're fine there.
12	Rely on your intuition and senses. It should feel good to be in the right carreer :-)
11	You are in the right carreer if you wake up every day and can't wait to get to work. You spend all day doing what you love and then when its time to go home you really don't want to.
8	What is the definition of the right? If people want, then they can do anything and enjoy doing it. Nobody preassigns you from the birth to be a cook or a janitor, it is a choice which is made based on the life experience and the environment.

The use of diploidy came from creative questioning method (Ray & Myers, 1989). It assumes the separation of messages into the two classes: problems and solutions. After such a separation, we can evaluate the fitness of each solution for a particular problem, not just evaluate if some message is a good idea in general (the case of GIB). FKE divides all processed text strings into the two mutually exclusive classes: problems and solutions, by analogy with female and male distinction. This distinction creates two levels of co-evolution in FKE, each having the same recombination methods, but different methods and criteria for selection. The interplay between problems and answers in the FKE create an effect similar to the effect of creative questioning method.

Here we draw several examples that were evolved in the system. Table 3 shows the question having the highest fitness at the time of writing this paper. Answers are ordered according to their fitness which is shown in the first column of the table.

Table 4 shows another example clarifying the meaning of word "knowledge" in FKE. Questions about word definitions are common in FKE. One of the extentions of the projects suggested by its participants is to create a self-maintaining web dictionary evolving with the language itself.

Table 4: Word definition question

Q	What is knowledge?
4	Approximation of the outside world in our local observable vicinity. It is usually expressed in some alphabet of a limited size and doesn't approximate well beyond the local limits.
4	Knowledge is our personal extrapolation of information. Our minds take in information (or data) and spew out knowledge – even when we're wrong.
3	Knowledge is information valuable for us, that we gather, select and generalize throughout our life.
3	Something that keeps us from making the same errors twice.
3	Something very powerful and hard to attain. It is knowledge about things as they are or reality. With the correct knowledge almost everything is possible.

The selection process in FKE is delegated to its participants as in interactive genetic algorithms (Takagi, 1998), but processing of the individual evaluations is different. The system acts as a mechanism that collects, processes, and integrates the individual selections made by humans. We assume that humans are error-prone and consider them as unreliable classifiers, so the main purpose of the whole classification system is to minimize the overall error of classification. This purpose can be achieved by different decision-making mechanisms: ensemble averaging, arcing and boosting, or multi-stage classification (Kosorukoff, 2000b).

The selection of problems is performed according to their importance, based on expressed interest of participants in each particular problem. This measure of fitness based on the summed interest of all participants is used to include a problem into the generated web pages shown to people. This process happens in interfaces of HBGA, which generate the interactive WWW pages dynamically. Roulette wheel selection method is used for this purpose. In this way, the problem in which many people are interested will appear in the interfaces more frequently. The frequency of appearance of the particular problem in the interfaces and in dynamically generated WWW pages can be thought as a measure of attention the system pays to a particular problem.

The selection of solutions is performed according to their fitness in the context of specific problem. The method of cascading classification used for this pur-

Table 5: Self-awareness question

Q	What is a goal of FKE?
10	Allow people to cooperate effectively, and optimize the technologies of their interaction.
8	To help people
4	Help every member to achieve his/her goal, succeed in his/her enterprise no matter commercial or non-commercial. Success of every member makes our community more successful, and expands opportunities of other members.
4	Attract many people, provide them with effective technology of creative cooperation, test new ideas, develop and implement them, evolve fast to satisfy continuously changing demands of participants.
8	Attract people and increase creative potential of their community.

pose is based on creating an optimal classification structure from individual elements and letting solutions propagate through this structure. The method of structure assembly described in details elsewhere (Kosorukoff, 2000b) is based on evolving the representations of classifying networks with a genetic algorithm to achieve the minimum of the overall classification error.

The interesting thing about the FKE system is that it can define its identity, purpose, and evaluate its own performance, evolving the answers to the corresponding questions: what is FKE? what is the purpose of this community of people? is it uselful? what is needed to make it better? By collecting this information the FKE system becomes 'aware' of what people think about it and which changes and improvements are needed. Most of these self-awareness questions appeared spontaneously in the process of evolution, as was the one shown in table 5.

These questions are circulating through the system, because participants express an interest to them. In this process the questions gather human opinions and evaluations, making the system aware of its purpose. Another self-evaluating question had the second best fitness at the moment of writing this paper. It is shown in table 6 with a list of the top 5 responses.

It this way, the FKE system becomes aware of its own performance. The satisfaction of people using the system is not a quantitative metric, but it agrees very well with the idea of this social system made for people. It can be said that FKE has no definite purpose. Human participants fill the purpose of FKE with their concerns and problems, and as long as these problems find solutions, the purpose of the whole organization is also fullfilled. To paraphrase Lao Tsu, FKE "has no purpose, but its purpose is fullfilled" (Tsu, 1972).

Table 6: Self-evaluation question

Q	What is your impression from this website?
10	It's very unique and really a good thing. This way people won't be afraid to ask.
9	This is a good way to continuously stimulate the thinking brain matters to keep one mentally fit
9	Could be helpful
8	The idea is quite smart. Here's hoping it can succeed, it's certainly got the potential
8	Interesting

Table 7: Correspondence between elements and procedures of GA and FKE

GA	**FKE**
Gene	Word of natural language
Chromosome	Text of question/answer
Population	Knowledge base
Fitness function	Human preference
Generation	Meta-interface cycle for solving a set of problems
Initialization	Solicitation of initial answers and migration of them from other populations
Selection	Ideas competition
Crossover	Crossover of answers Crossover of problems
Mutation	Random creativity technique

We still need to learn much about the mechanisms of evolutionary knowledge creation. The FKE project provides us with valuable data for this purpose: statistics about preferences of different people, methods of recombination and evaluation of natural language strings. What is clear by now is that FKE interfaces allow co-evolution of related populations of problems and solutions and this evolution results in selection of creative solutions and problems of interest. We believe that this is a sure way to new knowledge and understanding.

The correspondence between the FKE project and a GA is outlined in table 7. This table looks similar to the one for TQE. These are the same processes working in a different context. Different levels of evolu-

tionary process are emphasized in these two models. Comparing TQE and FKE, we can see that the former model represents better the processes in the higher levels (group and participating individuals), while the latter pays more attention to the lower levels (individual problems and finding solutions to them). Nevertheless the very same methods work on these levels to achieve the goal of quality of education (TQE) and effective creative problem solving (FKE).

Additional information about FKE and HBGAs can be found elsewhere (Kosorukoff, 2000a; Kosorukoff, 2001), but for the purpose of this paper the important thing is that despite major representation and implementation differences, the core concepts of FKE are the same as those of early GAs and most of the theoretical concepts of GAs are applicable to the processes of knowledge evolution in FKE.

4 EVOLUTIONARY ORGANIZATION

In this section, we present the two projects described earlier as examples of a single *evolutionary organization process*. We identify its structure and components and try to find the areas where it has advantages over more traditional forms of organization.

Our case study has shown how similar the processes behind TQE and FKE are to the main concepts of genetic algorithms and evolutionary computation in general. This similarity makes the two projects like-minded, so we can view them as two social applications of evolutionary computation. However, although these projects use the principles of evolutionary computation, we cannot call them strictly computational, since they use human intelligence as part of them. We suggest the term *evolutionary organization process* to reflect the hybrid fusion of computational and human efforts.

Examples of the evolutionary organization processes that we considered so far had their own meta-structure. We call it meta-structure to distinguish from the structure of organization that is created. We separate meta-structure into three components: innovation (mutation and recombination), selection, and organization. Each component can be computational or human-based. TQE is an example of a process where all components are human-based. In FKE organizational component is computational, while innovation and selection components are human-based. In processes such as FKE, where both computation and human-based components are present, interfaces between them become an important part of computational component. The meta-structures of TQE and FKE are shown in table 8 and 9 respectively. Innovation and selection components can be represented by multiple agents, or their parts.

Table 8: Meta-structure of TQE

Organization component (human-based)	
Innovation component (human-based)	Selection component (human-based)

Table 9: Meta-structure of FKE

Organization component (computational)	
Human-computer innovation interface	Human-computer selection interface
Innovation component (human-based)	Selection component (human-based)

If we imagine the situation where organizational, innovation, and selection components are all computational, we will get a typical GA. This suggests that we can apply knowledge about the design of effective GAs to the engineering of the organizational component, which controls major parameters of the evolutionary process (such as selection pressure and probabilities of different kinds of innovation). However, we should always keep in mind the difference between the goals of evolution in a typical EC application and in evolutionary organization process: in the former case, the goal is fast convergence to the optimal design; in the latter case, the goal is an on-going process of innovation that should never get to a halt.

Now it is time to go from implementation to the results and discuss the actual system of organization that we get with evolutionary approach. Table 10 compares a traditional system organization with an evolutionary one.

Traditional organization is characterized by a structure that does not change often. The structure of organization is defined by its lines of communication. Usually the structure of communication is defined by rules, for example, “in case of computer failure call computer maintenance department”. As with a mechanical device, traditional organization assumes that each part will perform its function. In our case, a person in the computer maintenance department will answer the call and handle the problem. In this case, the communication process follows a fixed pattern or structure. Most violations of this pattern are harmful for the system as

Table 10: Comparison of traditional and evolutionary system of organization

Traditional organization	**Evolutionary organization**
Fixed form	Free form
Pre-designed based on static assumptions	Emerging and changing in the process: no apriori assumptions
Hard parts	Soft parts
Functional design: each part performs a specific pre-defined function	Organic design: multi-functional parts, not confined to performing a particular function
Emphasizes structure	Emphasizes process
Obligatory: based on contract	Participatory: based on contribution
Error correction: rejects spontaneous changes	Error utilization: uses spontaneous changes
Correctness based fitness	Achievement based fitness

a whole. For example, the failure of some part to perform its function often leads to the failure of the whole organization or at least some large part of the organization. Routing the problem to the wrong specialized unit will also result in functional failure. In these circumstances, reliability of parts and adherence to the established structure becomes the major priority.

When we consider an evolutionary organization, we notice many contrasts to the traditional one. There is no fixed structure. For example, we can not determine to which participant a particular problem will be routed in the FKE project. No matter who this person will be, his/her failure to solve the problem does not mean the failure of the whole system. Instead of a fixed structure of interactions, we have a stochastic process that recreates the new structure each time when communication is needed. It is easy to notice that in this case the relationship between structure and process is the opposite. Parts of the organizations are rather organic than functional, in other words they have ability to perform different functions in different time.

Since evolutionary organization process is built after the principles of EC, we can use the design principles of effective GAs to evolutionary organizations. However, we should keep in mind that the goal of evolutionary organization is different from the goal of standard GA. It is clear that fast convergence is not a goal of evolutionary organization. Convergence in the case of FKE would mean leaving a participant without any freedom of choice, and insisting that one 'correct' answer will work for the problem, in the case of TQE convergence would mean that all teams lose their identity, in the pursuit of perfection. That is not what we want in social environment. In this case any convergence will be premature, because the real-world does not stop there. A living organism which stops to adapt to changes will eventually die no matter how perfect it was before unless someone will take care of it. Convergence has little meaning in the living world, that is why traditional metrics of quality of the genetic algorithm based on time-complexity are often inadequate in these cases. For evolutionary organization we need an on-going evolutionary process, one that adapts easily to the changes of environment, never converging to the current best solution.

The experience with evolutionary organization process shows that the type of EC must correspond to the area of its application. Technical areas need competent GAs. Social areas need balanced or enlightened GAs (Goldberg, 1989). While competent GAs are designed to achieve fast convergence, balanced GAs can be designed to achieve innovation and creativity as a continuing process. Balanced GAs should be able to adapt to always changing environment, they should check if their assumptions about the world are still valid, and should be able to 'unlearn' them easily if they are not.

5 SUMMARY

The paper has considered two social systems designed explicitly with a GA in mind. The first of these organizes an educational process efficiently. The second promotes collaborative problem solving on the web. In both cases the inspiration and connection to GAs is clear. In the first, all functions of the system were performed by human beings. In the second, a mix of computational infrastructure and human interaction worked together. In both cases, interesting system behavior was observed and is continuing.

The paper continued by generalizing these types of systems and by calling such combines of human and computational power, evolutionary organization processes. The meta-structure of social organization processes was developed and more generally social organization processes were contrasted with traditional organizations.

While many if not most of the attendees of this conference are pleased to solve their organization's technical problems using the latest in genetic algorithms and evolutionary computation, this paper suggests that we may all have a larger role to play in the solution of

our organization's organizational problems by appealing to our GAs and EC for inspiration, specific structure, and even population parameter settings. There is much work to do, but we believe that the examples and framework of this paper may be useful to these future efforts.

Acknowledgments

The authors would like to thank Jay Mittenthal and Chi-Sheng Ho for interesting discussions and contributing valuable ideas to this paper. We are also thankful to the participants of the Free Knowledge Exchange project contributed the material included in this paper and granted the right to publish it under the conditions of the Success Formula Community Agreement.

The first author was supported by funds from the Technology Entrepreneur Center at the University of Illinois and that support is gratefully acknowledged. Professor Goldberg's contribution to this work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-97-1-0050 and F49620-00-0163, and by the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the researchers and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

References

Bandler, R., & Grinder, J. (1976). *The structure of magic: A book about language and therapy.* Palo Alto, Calif.: Science & Behavior Books.

Dawkins, R. (1976). *The selfish gene.* Oxford: Oxford University Press.

Eno, B. (1998). *An incremental strategy for changing the world* (Foreword to Global Ideas Bank Compendium). GIB. www.globalideasbank.org /wbi/WBI-1.HTML.

Goldberg, D. E. (1983). *Computer-aided gas pipeline operation using genetic algorithms and rule learning.* PhD dissertation, University of Michigan.

Goldberg, D. E. (1989). Zen and the art of genetic algorithms. *International Conference on Genetic Algorithms '89*, 80–85. (Also TCGA Report 88003).

Goldberg, D. E. (2000). The design of innovation: Lessons from genetic algorithms, lessons for the real world. *Technological Forecasting and Social Change*, *64*, 7–12.

Goldberg, D. E., Hall, W. B., Krussow, L., Lee, E., & Walker, A. (1998). *Teamwork for Quality Education: Low-cost, effective educational reform, through a department-wide competition of teams* (IlliGAL Report 98005). Department of General Engineering, UIUC.

Herdy, M. (1996). Evolution strategies with subjective selection. In *Parallel Problem Solving from Nature, PPSN IV*, Volume 1141 of *LNCS* (pp. 22–31).

Holland, J. H. (1995). *Hidden order: How adaptation builds complexity.* New York: Addison-Wesley.

Kosorukoff, A. (2000a). Human Based Genetic Algorithm (HBGA). `http://3form.com/hbga`.

Kosorukoff, A. (2000b). Social classification structures. optimal decision making in an organization. In *Late breaking papers of GECCO'2000*, Genetic and Evolutionary Computation Conference Proceedings (pp. 175–178). Las Vegas, Nevada.

Kosorukoff, A. (2001). Human based genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC–2001*, 3464–3469.

Pinker, S. (1998). *How the mind works.* W. W. Norton.

Ray, M., & Myers, R. (1989). *Creativity in business.* New York: Doubleday.

Takagi, H. (1998). Interactive evolutionary computation: System optimization based on human subjective evaluation. *INES'98*.

Tsu, L. (1972). *Tao te ching (Gia-fu Feng & Jane English trans.).* Wildwood House.

The Turing Ratio: Metrics for Open-Ended Tasks

Hassan Masum, Steffen Christensen and Franz Oppacher
Department of Computer Science , Carleton University
1125 Colonel By Drive, Ottawa, Canada K1S 5B6
{hmasum, schriste, oppacher}@scs.carleton.ca

Abstract

The Turing Test is of limited use for programs falling far short of human performance levels. We suggest an extension of Turing's idea to a more differentiated measure - the "Turing Ratio" - which provides a framework for comparing both human and algorithmic task performance. Games provide an example of the concepts. It is argued that evolutionary computation is a key method for amplifying human intelligence, an assertion which future scientists can empirically decide through measuring Turing Ratios and considering task breadth, prior knowledge, and time series of the measures.

1 INTRODUCTION

1.1 MEASURING TASK PERFORMANCE

Boundaries between computational and human intelligence will blur given sufficient algorithmic advances. Hence we may wish to rank computational performance directly against humans, or against an objective standard if possible; this allows comparisons and time series extrapolations to be made objectively.

However, task performance for open-ended tasks (like Turing's suggestion of unrestricted conversation with a human) has not heretofore been sufficiently formalized, in a manner allowing easy comparison across task domains and time scales. As well, no program is close to being ready to take a broad-based intelligence test.

1.2 THE TURING RATIO

As a solution, we propose a framework: measure the performance of computational entities on well-defined tasks, and compare these performances with each other and with human performance. This will provide an objective and reproducible measure of "intellectual ability", upon which further analysis can be based.

Once many domains have been assigned Turing Ratios, one might rank the domains by increasing Ratio to quantify the "automatizability" of different kinds of tasks. This will tell us something about the intrinsic difficulty of the domain - perhaps it will confirm our current intuition about what tasks are difficult, or perhaps there will be surprises in store.

With human and computer performance measured on the same scale, one may compare economic value. Due to Moore's Law and algorithmic advancement, trends of Turing Ratios may suggest tasks which will become economically unviable for humans, and conversely point out situations currently overloading human cognitive limits that may be alleviated via algorithmic assistance.

We discuss refinements to this basic idea: the amount of prior knowledge used by a program gives insight into the robustness and domain-specific customization currently necessary for good performance in the given domain. Learning from scratch is more impressive than being thoroughly customized for a domain, and this is a strength of Evolutionary Computation (EC) that is quantifiable in principle. Breadth of the domain in which good performance has been achieved is also important; one practical definition of "intelligence" is the ability to survive in diverse environments.

Turing Ratios can quantify empirical EC progress, generate objective data for inference about the algorithmic properties of hard tasks, and suggest areas of human uniqueness.

1.3 RELATED WORK

In [Koza 1999], domains are listed in which computers have outperformed humans who tried the same problems. As well, benchmarks for ranking GP results as human-equivalent in a domain are suggested; most are variants of the idea that a reputable publication, patent, or competition result should be regarded as valid evidence of achievement regardless of the source.

The Turing Test was first proposed by Alan Turing in [Turing 1950]. By providing an operational benchmark which could plausibly be passed by an entity if and only if that entity was in some sense human, Turing moved the discussion of artificial intelligence to a new and more productive plane. Many variants have been proposed; the extensive discussion in the half century since Turing's seminal paper is reviewed in [French 2000]

As a signal indicating the presence of strong AI, the Turing

Test is arguably as good as any other single measure yet devised. However, it has the drawback of being relatively binary; there is no provision for partial success, and hence the Test may not be of use until far in the future when researchers have neared the goal of creating human-equivalent AI. (See [Saygin et al 2000] for further discussion of the Test and its instantiation in the Loebner Prize competition.)

2 TURING RATIO THEORY

2.1 ASSUMPTIONS AND SUITABLE TASKS

No single-dimensional measurement can adequately capture all the varieties of "intelligence". Hence, we restrict our claim for the validity of the Turing Ratio (TR) to measurement of task performance. Some key assumptions are necessary:

- Task definition and performance are reproducible and unambiguously defined.
- Task-solving entity is clearly delineated.
- Finally, task performance should be adequately represented with a single-dimensional variable.

Although the single-dimensional assumption may seem restrictive for open-ended tasks, the original Turing Test manages to cleverly reduce conversational performance to a single variable: number of judges who agree that the speaker is human. (For multiobjective tasks with "stable subobjectives", the subobjectives could be treated independently. Extension to complex multiobjective cases is deferred for future research.)

Coming up with tasks that encapsulate other interesting domains is a challenge (the "progress parameterization problem") akin to finding the right fitness function for a complex EC instance. This suggests that task measurements for domains where quantizing performance is non-obvious might be derived from fitness functions for EC programs for those domains, and vice versa.

In addition to the necessary conditions above, we suggest three properties a task should have for the Turing Ratio to be a suitable measure on it:

1. Task measures an "important" ability.
2. Task admits a series of increasingly better strategies, which are qualitatively different.
3. Reproducible and relevant for decades.

To the degree that a particular task requires intelligent solution methods, a high TR value for that task indicates the presence of intelligence.

2.2 SINGLE AND MULTIPLE MEASUREMENTS

There are four basic types of single-dimensional variable: nominal, ordinal, interval, and ratio. These correspond to different types of TR measurement tasks:

- Nominal: A single contest of the Turing Test - where a judge rates a conversant as human or not - could be viewed as a single nominal-valued measurement.
- Ordinal: Games are the prototypical task where ordinal measurements are taken, as discussed later.
- Interval: Ideal for task measurements without a known absolute reference point, such as many human test scores.
- Ratio: Ratio scales give the most information, possessing an absolute reference point. Such measurements are commonly used in computational complexity, where the time or space requirements for an algorithm are precisely defined.

What happens with repeated task measurements? In the context of TR, statistics confer two main benefits: the reduction of uncertainty, and synthetic generation of more detailed task measurements. The first benefit is widely understood, e.g. multiple Turing Test judges induce a binomial random variable with associated confidence intervals; [Jain 1991] discusses similar "classic" performance measurement.

The second benefit of synthetic derived variables is easily understood in the context of games. Given a particular ordinal task measurement - say a win or loss against a game opponent - we can infer only a limited amount about underlying abilities. However, repeated trials allow an increasingly precise demarcation to be made between various ability levels, generating a synthetic interval or ratio scale - an issue we discuss further in our chess example. For competitive situations like the game example, it's important to note that repeated single-dimensional measurements are limited in the resolution of the derived synthetic variable by the number of measurements, the variability of the task performer, and (to the degree that abilities are not totally ordered) the distribution of opponents.

2.3 ABSOLUTE AND RELATIVE TR

We distinguish absolute and relative performance. The intuition is that a well-defined algorithmic task that can be measured in isolation is an absolute task, while a game against opponents is a relative task depending on the current opponent ability distribution. (One could make an analogy to evolution in a static environment versus coevolution.)

An Absolute Turing Ratio is therefore defined as a TR independent of when the result was achieved, the performance levels of others, and the competence level of the performance evaluation past a given threshold requirement. Note that this is a strong requirement: most open-ended tasks rely directly or indirectly upon fitness comparisons between competitors. What would be required is a completely specified task, measuring important abilities, valid for decades, admitting a series of increasingly better (and qualitatively different) strategies, where the task is performed without reference to other competitors and evaluated algorithmically. We regard it as an open challenge to specify such tasks. (Ideally, they would be "human-complete", in analogy with NP-complete tasks.

Turing's task of conversation imitation, for example, is arguably rich enough to be a sufficient indicator for human-level thought. It is not an Absolute task, though, since the worth of the evaluation depends strongly on the competence and insight of the human judges.)

In contrast, a Relative Turing Ratio result needs to specify the opponent distribution along with the environment. If the task is widespread enough, it may be sufficient to simply specify the time period in question, e.g. the chess environment circa 2000. (If Absolute versus Relative is not specified, one can assume Relative. This is due to the current lack of well-specified Absolute task domains - a key area to be formalized.)

Usually a given algorithm will vary its performance depending on the resources available to it: time and space are the two typical ones. More generally, having access to an "oracle" that performs a specified black-box function may make a substantial performance difference, e.g. access to the Internet. Thus, TR ratings should specify CPU, memory, and any salient modules or prior information incorporated.

We note that the requirements of resource and opponent specification are consequences of the original assumption that task performance be unambiguously reproducible. This is essential for systematic comparison between problem domains, solution methods, and task performance over time.

2.4 FORMAL SPECIFICATION

Let's look at two examples of formally specifying TR. First, consider the general specification. Given a program p for a specified task domain D (where p belongs to P, the population of competitors), the computational performance of p in D will be denoted by $\Phi(p, D)$. (Φ returns one of the four variable types defined previously.) We are then interested in measuring:

$$\Phi(p, D, TimeStamp, Resources, P)$$

Resources could be any set of computationally-useful resources; if one or more of these resources can be parameterized by a scalar variable, we can do some informative statistical extrapolation, as we'll see. (P would not need to be specified for an Absolute Turing Ratio result.) One could also include additional information in a TR measurement; the underlying idea is that a well-specified minimal set of information should always be present, so that long-term analysis can usefully be accomplished, including perhaps reparameterization of TR based on a user's judgement.

In the case of n repeated trials, we get:

$$\Phi^n(p, D, Resources, ...)$$

i.e. Φ repeated n times. If Φ is a nominal variable, then Φ^n will be drawn from a binomial or multinomial distribution; in the binomial case, we can use a Beta distribution to infer the probabilities of Φ taking each possible value.

Often, we are interested in comparing two performances (either of distinct programs or of the same program with different resources):

$$\Phi_{Relative}(\Phi(p_1), \Phi(p_2))$$

This should be a function returning a meaningful "ratio": three empirically-common options are simple division (in the ratio-variable case), differences in log performance (so a constant increase in TR implies a constant multiplicative factor increase in Φ), and percentile differences (i.e. relative population ranking).

Finally, we note that the spatial distribution of TR scores could be of interest; this can be modeled by placing competitors in some graph, and measuring TR ratios relative to other scores within some distance.

Now consider a specific program and task. We measure:

- $CompScore = \Phi(p, D)$.
- $MaxCompScore$ = max $\Phi(p_i, D)$ over all programs p_i in D.
- $RefHumanScore$ = e.g. max or median $\Phi(Human_i, D)$ over all human performances in D.

Then we can define the program's TR, and similarly a domain-wide TR for the best computational performance:

$$TR(p, D) = \Phi_{Relative}(CompScore, RefHumanScore)$$
$$TR(D) = \Phi_{Relative}(MaxCompScore, RefHumanScore)$$

An actual TR measurement would contain more detail - enough to unambiguously specify and reproduce the result, including particularly the domain, resources, and program. The construction of $\Phi_{Relative}$ may also change over time, depending on the opponent distribution and the meaning of a particular score.

We can divide task domains into three broad categories by their domain-wide TR: TR(D) << 0 dB (computer much worse than human), TR(D) >> 0 dB (computer much better than human), and TR(D) ~ 0 dB (rough equality). The interesting domains at any point in time will be those on the ever-expanding "Turing Frontier", where algorithmic task performance is improving rapidly relative to human performance.

3 TURING RATIO EXAMPLES

3.1 A CASE STUDY: CHESS RATINGS

Consider two agents A and B playing a stochastic game involving personal talent against one another. Let us label the talent levels of A and B as a and b. A simple model sets the win expectancy for player A as:

$$p_{win}(A) = a/(a + b)$$

If we take the ratio of the ratings of two individuals playing, we can let ρ be a/b, and we get the single-parameter form:

$$p_{win}(A) = \rho / (1 + \rho); \; p_{win}(B) = 1 / (1 + \rho)$$

Intuition suggests a logarithmic scale would be best to

evaluate the wide range of abilities in most domains; one candidate is the decibel scale commonly used to measure signal strength. We therefore define:

$$TR \text{ (in dB)} = 10 \log_{10} (\rho_{measured}) - ReferenceAbility$$

We suggest the convention of letting the median human performance in a domain, where ascertainable, be given the reference level 0 dB. *CompScore* and *MaxCompScore* are then the empirically measured TR values of the current and best program respectively. This Turing Ratio scale represents each person's performance as a real-valued parameter. (The dB unit is easy to understand; however, TR could be quoted in other units.)

Note that deriving the parameter ρ from observed wins and losses can be a complex procedure. We are reducing a two-dimensional matrix of observed wins and losses to a single-dimensional measure; this can only be done in a reasonable way if the original matrix is well-behaved (e.g. there are not too many transitive winner cycles A-B-C-A). Even if the measure can be found, it may be only approximate due to measurement and performance errors, performance nonstationarity, and so on. We note that a practical implementation of the Turing Ratio would need to consider all these issues, and refer the interested reader to [Glickman 1999] for one scheme to infer ratings from large competitions.

Our TR scale is not without precedent; there is a close relationship with the United States Chess Federation rating system. We outline the similarities and show how chess programs and humans can be effectively rated on the same scale.

The USCF rating system rates a decimal order of magnitude performance difference at 400 rating points [USCF 2002]. Therefore:

$$USCF\ rating = 40(dB\ rating) + HumanMedian$$

where *HumanMedian* is median human chess performance, around 600 USCF rating points. We can even use the USCF rules directly to measure performance in any pairwise competitive endeavour where the possible outcomes are Win/Lose/Draw [USCF 2002].

In addition, the Swedish Computer Chess Association (SSDF) has approximate empirical scaling laws for size of the in-memory hash tables used in computer chess programs, and for performance at chess as a function of processor speed. (Note that the SSDF calibrates its point scheme against the World Chess Federation (FIDE) system, which uses a Gaussian distribution scheme with a slightly larger spread then the USCF. In FIDE, there are roughly 410-points in an order-of-magnitude performance difference[1].

As quoted in the SSDF FAQ [Grottling 1998], Kathe Spracklen estimated the effect of doubling the in-memory hash tables at 7 ratings points, or a performance factor of $10^{7/410} = 1.04$, about a 4% improvement in relative Turing Ratio. The same FAQ also estimates the effect of doubling processor speed at 70 ratings points, for a performance factor of $10^{70/400} = 1.48$, or a 48% improvement.

We thus have a practical case where both computer and human performance have been compared on the same scale, in a Turing Ratio framework.

3.2 POTENTIAL METRICS AND TASKS

A canonical metric is time to complete task, or size of task that can be completed, for e.g. combinatorial and NP-hard problems. More generally, computational complexity theory suggests many metrics measuring aspects of problem hardness.

Games are particularly good domains due to unambiguous task scope and "complex strategies from simple rules". Checkers and chess both have computer players rated higher than any human since the 1990's, with Schaeffer's Chinook victories and Deep Blue respectively. Go programs are still well short of the best humans. Go has a well-defined rating system, and algorithms have slowly been improving in rank; [Bouzy 2001] discusses the algorithmic challenges. (Note that since human reference levels change slowly with respect to computer performance, we might take them to be fixed as a first approximation. Also, we certainly do not claim strategy games are fully open-ended tasks, but relative to their simplicity of definition they do admit a vast series of increasingly creative strategies.)

An important human rating method is tests of competence and "practical substitution": put someone in a situation and see how well they perform. Exams, aptitude tests, and IQ tests - usually relative since they are scaled to student body ability - attempt to estimate likely performance and measure current competence. (They have been widely criticized for measuring only part of "intelligence". Reduction to a single parameter probably cannot summarize a complex mind's performance - the same will be true of computational performance in broad enough domains.)

One might quantify human judgement of performance, e.g. ranking in a music competition, money earned, market or attention share, double-blind human comparison, or votes by an audience. Clearly literature, art and music are among many such tasks with very low TR at present.

Although many computational testbed tasks are intellectual in nature, we believe TR should also be applied to tasks that are embedded in the physical world. Two practical examples of such tasks are driving in real-world conditions, and building a cockroach-killing robot; both these would fulfil very practical needs while demonstrating a wide range of competencies, and will likely admit slowly-increasing performance for decades.

[1] From a Levenberg-Marquardt fit for A in the function $p_{win} = 1/(10^{-\Delta R/A} + 1)$ to the FIDE win expectation curve for rating differences from –100 to 100 [FIDE 2000].

4 BOOTSTRAPPING INTELLIGENCE

What does a given TR value mean? One of our main goals is to define a simple operational measure, from which characteristics of performance at open-ended tasks can be empirically derived. For such analysis, one needs to consider factors in addition to TR value. One key factor is the amount of "bootstrapping" provided by the algorithm; others include normalizing the TR-value achieved by breadth of task scope, computing power available, or human assistance.

4.1 INTELLIGENCE AMPLIFICATION

A game player may not care which method was used to generate an excellent Go-playing program, or whether it achieves its results through a clever billion-move opening book and straightforward many-ply search. However, an algorithm designer may value methods using a minimum of prior knowledge more highly, both for conceptual reasons and for their greater likelihood of rapid future advancement and robustness.

How impressively a program performs depends not just on its performance but also on how much prior customization and problem-specific hardwiring went into it. A program that has millions of moves or answers prestored by human hand may be regarded as correspondingly less impressive, regardless of its TR-value - this is why Fogel's evolved checkers player in [Chellapilla & Fogel 2000] is as significant an accomplishment as Schaeffer's hand-tuned Chinook, even though the latter won against the human world champion (as recounted in [Schaeffer 1997]).

To some extent, these opposing goals represent a task-centered versus a program-centered point of view. In the former, the main goal is to understand computational performance levels in a certain task, with the specific computational methods (and even to some extent the cost or time investment involved) being a secondary concern. Such a point of view may be suitable for assessing trends in performance levels over time, for understanding the automatizability of a task or cognitive domain, or for understanding the state of the art. In contrast, a program-centered point of view also considers the "intelligence amplification" aspect: how the program achieved its performance level is as important as what that performance level turns out to be. This point of view is appropriate to those trying to understand the relative adaptability of various computational methods.

At one extreme, a program has a full list of answers given to it; at the other extreme it starts with zero domain knowledge, and succeeds in learning good solutions. In between, there are various degrees of "cooking": exhaustive enumeration of cases, hand-designed primitive functions, intermediate reinforcements, externally supplied fitness functions, and so forth. This distinction is analogous to that between rote and deep learning: in the former all knowledge is given by the teacher, while in the latter the teacher suggests insights and poses problems but the student does most of the problem-solving (and hence develops better metastrategies and metaphors which increase general learning power).

With tongue somewhat in cheek, we suggest the terms "cooked AI" and "raw AI" to name these distinctions. Cooked AI has a high degree of problem-specific heuristics, expert knowledge, and fixed representation; raw AI is more readily generalizable, and autonomously learns. Cooked AI solutions may be good for many domains where one does not mind spending the time and money to customize a solution; raw AI solutions (e.g. EC) tend to learn or bootstrap more during their operational phase, and be cheaper computationally and financially - and hence likely more flexible if the domain changes.

How could intelligence amplification be measured? It would be a way of measuring the distance between a program's starting state of knowledge and the results it eventually achieves. One method might be to measure the increase in Turing Ratio as the program adapts to and learns its computational task:

TR(final high-performance state) - TR(start state)

This difference would objectively measure the improvement in performance of the program. Of course, in many programs this iterative improvement occurs via human programmers providing the improvement, so we look specifically for "autonomous increase in TR", i.e. that due to the program itself once completed and running. By this measure, Chinook shows a much smaller increase in TR than Fogel's checkers player.

A more practical definition may be to scale TR by the cost of different approaches. A heavily customized expert system typically requires the combined efforts of many programmers and knowledge engineers working for years to achieve adequate performance. In contrast, EC approaches have often achieved comparable performance with substantially less effort. One might empirically measure:

$$Cost_{Approach} = Cost_{Programming} + Cost_{Processing}$$

We hypothesize that this measure will tend to be lower for methods with higher intelligence amplification.

Computational depth may suggest formalizations of intelligence amplification. Intuitively, an object is computationally deep if it has a short generating program but that program takes a long time to produce its output. Perhaps the benefits of interaction with a rich environment could be algorithmically described as access to a rich store of precomputed computationally deep objects. The basic definition of computational depth using Kolmogorov Complexity is in [Antunes et al 2001], but the ramifications of this idea remain to be explored.

Explicitly measuring prior knowledge and intelligence amplification may become a nonissue in the long run, since approaches which rely on massive pre-knowledge or

customization for each problem may be increasingly difficult to program for more human-complete problems - and hence the easier-to-program methods will naturally win out over time.

4.2 TASK BREADTH

Many customized programs work well in a narrow domain, but have little or no computational capability elsewhere. Whether this is an issue depends on the user - a task-centered user wanting only a black-box resource to solve subproblem X or an analysis of the level of best current computer performance at the task will care little about what else the black box can and cannot do, while a program-centered user analyzing the intrinsic merit of the heuristics used by the program may care a great deal. (Note though that even a task-centered user may be wary of low-intelligence-amplification programs due to potential brittleness and high knowledge capture costs.)

It would be nice to be able to define a notion of task breadth in an objective way, so that one could speak of e.g. a program's high TR ratio on a narrow-scope task, versus a lower TR on a larger-scope task. One could then compare different methods and heuristics for their breadth-scaled TR. Parameterization of tasks by breadth is also essential in comparing task areas, and perhaps in forming an "ontology of task space".

A definition of task breadth could give an alternative definition of intelligence amplification as "degree of adaptiveness": the breadth of tasks for which the program performs adequately, given initial adequate performance on a limited task. Perhaps an objective and operational measure of intelligence itself could also be derived, as adaptiveness on a sufficiently-diverse environment set.

If we think about measuring the Turing Ratio of human-complete tasks in the near future, values will likely be close to zero for quite a while. Hence we could measure competence on a very restricted subset (which would give reasonably differentiated values), competence on the full set (which would be very low, e.g. the Loebner prize competition as discussed in [Saygin et al 2000]), or finally competence in some series of successively more difficult tasks between what is achievable now and the full task. This last option might be best, but would require a nested series of tasks with successively higher task breadth; one might expect to see a series of logistic curves on the task series, which would need to be chosen at the right difficulty level to be doable given competence at the previous task (similar to teaching a human, and to how solutions for a tough EC task might be evolved in stages).

4.3 DEGREE OF HUMAN ASSISTANCE

We can distinguish three general cases:

1. Isolated algorithm. Once the task is defined and the program complete, the solution process is autonomous. (The border between this category and the next is fuzzy, e.g. human selection of "interesting output" from evolutionary art programs.)
2. Human-algorithm symbiosis. A human is involved in significant parts of the computational process.
3. Isolated human. The "base case" from which our historical and social experience before the 20th century is derived.

We have been comparing the first and third cases; the second may also be of interest for certain applications, in particular those for which

$$TR(algorithm+human) - TR(algorithm)$$

is large. As with intelligence amplification, for some purposes one might not care if superhuman performance is achieved by an autonomous program or by a human-computer symbiosis; for instance, one could picture a Go grandmaster using a Go AI program to explore possible moves, and beating an unassisted grandmaster of equivalent ability. For other purposes such as program analysis or identification of areas of future rapid progress in performance or cost savings, it would be important to maintain the distinction.

5 USAGE

5.1 GAMES AND TIME SERIES

Games would be an excellent testbed for the Turing Ratio. They are well-specified domains, almost always with clear conditions for winning and losing, and are a domain of much interest among both researchers and the public at large. Scoring systems have been developed for many games, and a great deal of knowledge and experience (both human and computational) has been gained in the more popular human games. Many AI systems have been developed; see e.g. the first part of [Bouzy 2001]. Note too that handicapping allows meaningful ranking comparison between a human and a program with superhuman performance.

Games are usually multiplayer. One-person "games" like combinatorial optimization problems (or even solitaire games) seem less open-ended. In contrast, multiplayer games specify an "interaction protocol" which (for good games) can be played with increasingly higher sophistication by players that keep learning. Hence multiplayer games encourage a coevolutionary increase in strategy sophistication; see [Funes 2001] for an example where Tron agents were evolved using GP in competition with human players.

Time series of ratings will give tools for visual and trend analysis. The statistical significance of a Turing Ratio result for a particular domain (e.g. the variance or stationarity of a particular TR rating) may be an issue in some cases; empirically, however, humans do still get ranked in many domains. For many tasks, improvements in TR will probably

come in large discrete jumps (e.g. when a clever data structure reduces the algorithm's time complexity by an order of magnitude). We hypothesize that tasks which are more open-ended and creative will have more and smaller jumps, leading perhaps to more predictability in the rate of improvement of such problems (via the Central Limit Theorem).

Similarly, it may be possible to derive taxonomies of domains by the current Turing Ratio. Looking at relative computational strengths between different types of cognitive tasks in some sense tells us how well we understand those tasks, since being able to build an algorithm to perform a task is a strong form of knowledge. As Richard Feynman said: "What I cannot build, I do not understand."

5.2 RESOURCE-SCALED TR

We can use a "resource-bounded Turing Ratio" by bounding computational resources (time, space, etc), and comparing competing algorithms by their rating under the given resource bounds. This factors out improvement due solely to resource differentials.

Complementarily, TR scales with increasing computational power. The inverse of the time required for adequate performance tells how the size of problem scales with resources; more subtly, the graph of TR achieved versus processor power may have a variety of behaviors, the qualitative categories of which form another way of characterizing difficult domains

A low average slope would indicate cases where throwing more hardware at the problem is insufficient to significantly increase performance - an example might be PSPACE-hard problems without good probabilistic approximation schemes. Perhaps in such cases EC methods improve slowly with computational power, and must pass through an inherently sequential “critical path” series of inferences as well as spend time interacting with the real world in order to come up with creative solutions.

Once there exist algorithms with sufficiently high TR for a domain, there may still be a human in the loop for part of the task (including pre or post processing). We can then divide the total time of the symbiotic human-computer team *SymbTime* into human and computer portions, and consider the long-term distribution of these portions. The continuing progression of Moore's Law implies that the cost for a given level of TR performance will decay exponentially for tasks whose improvement scales fast enough with increasing computational power, at least for as long as Moore's Law continues to hold. This in turn suggests the following definitions and ratios:

- $SymbTime = Time_{Comp} + Time_{Human}$
- *HumanTime* = unaided human time for task
- $HumanRatio = Time_{Human} / HumanTime$
- $CompRatio = Time_{Comp} / HumanTime$

$Time_{Comp}$ accounts for the portion which has been algorithmized, while $Time_{Human}$ is the portion needing humans to perform adequately. *HumanRatio* then represents the limiting reduced time that can be achieved as a fraction of the time required by an unaided human. This is so since the Moore’s Law assumption implies

$$Time_{Comp}(now+T) = 2^{-cT} * Time_{Comp}(now)$$

so *CompRatio* will decrease exponentially. (Although poor user interfaces could result in a *HumanRatio* greater than 1, the cases of greatest interest are cognitively complex tasks with *HumanRatio* close to 0, since they are the tasks which can be usefully outsourced from humans to computers.)

Note that this result also holds for other exponentially-increasing computational properties with which a problem’s TR scales well, such as memory. Although exponential growth of a computational property is a temporary phenomenon in a bounded universe, the fact that we are in the middle of several such temporary regimes makes the analysis relevant to the early 21st century. [Hanson JAIR] discusses further consequences of machine intelligence for economic growth.

Ranking domains by *HumanRatio* - and graphing values for domains over time - will give empirical evidence as to the cognitive tasks for which human thought is most suited and required, and the tasks best done by computers. Similarly, one could potentially objectively identify heuristics or approaches that caused large decreases in this ratio for particular domains, and perhaps see some "meta-patterns" in the types of approaches that give the most leverage.

6 CONCLUSIONS

We have demonstrated the formalization of algorithmic performance through the Turing Ratio, and conceptually validated it through empirical examination of chess program performance. Along with the companion notions of intelligence amplification and task breadth, the Turing Ratio forms a conceptual foundation for measuring long-term progress in evolutionary computation.

As [Bentley & Corne 2002] vividly show, evolutionary computation techniques are already producing many results that deserve the label “creative”. To the degree that progress in open-ended and creative tasks can be meaningfully quantified, increasing amounts of empirical performance results will become available for analysis.

Three key challenges now are to further develop the theory, identify suitable applications, and begin to measure TR values systematically. It is true that TR as we have defined it is applicable only to the subset of tasks satisfying the conditions of Section 2.1. However, we hypothesize that this subset consists of tasks which, once identified and explored, may serve as windows into creative algorithmic performance. With links to other areas of AI, performance measurement, and game playing, we believe this is fertile ground for future research.

Acknowledgments

Alan Turing inspired generations of computer scientists with his Turing Test and Turing Machine concepts. We hope he would have appreciated our attempt to extend his ideas.

Thanks to Alex Tulai, Ben Houston, and Andrew Vardy of Carleton University's ALEC seminar for a healthy trial by fire of some of these ideas. Valuable feedback was also received from several anonymous referees.

References

[Antunes et al 2001] Computational Depth. L Antunes, L Fortnow, and D van Melkebeek; in *Proceedings of the 16th IEEE Conference on Computational Complexity*, pp 266-273; IEEE, 2001.

[Bentley & Corne 2002] *Creative Evolutionary Systems*. Peter J Bentley and David Corne; Academic Press, 2002.

[Bouzy 2001] Computer Go: An AI-Oriented Survey. Bruno Bouzy; in *Artificial Intelligence* (Vol.132 No.1), pp 39-103.

[Chellapilla & Fogel 2000] Anaconda Defeats Hoyle 6-0: A Case Study Competing an Evolved Checkers Program against Commercially Available Software. Kumar Chellapilla and David B Fogel; in *Proceedings of the 2000 Congress on Evolutionary Computation*, pp 857-863; IEEE Press, 2000.

[FIDE 2000] "The Working of the FIDE rating system", 2000 Fédération Internationale d'Échecs (FIDE) Handbook, section B, 02.10.1a; viewed at http://handbook.fide.com on 14 March 2002.

[French 2000] The Turing Test: the first fifty years. Robert M French; in *Trends in Cognitive Sciences* (Vol.4 No.3), pp 115-121.

[Funes 2001] *Evolution of Complexity in Real-World Domains*. Pablo Funes; PhD Thesis, Brandeis University, 2001.

[Glickman 1999] Parameter estimation in large dynamic paired comparison experiments. Mark Glickman; in *Applied Statistics* (Vol.48), pp 377-394.

[Grottling 1998] *Frequently Asked Questions about the SSDF Rating List.* Göran Grottling, 1998; viewed at http://home.swipnet.se/~w-36794/ssdf/ssdf-faq.htm on 18 January 2002.

[Hanson JAIR] Economic Growth Given Machine Intelligence. Robin Hanson; to appear in *Journal of Artificial Intelligence Research*.

[Jain 1991] *The Art of Computer Systems Performance Analysis.* Raj Jain; John Wiley & Sons, 1991.

[Koza 1999] *Genetic Programming III*. John R Koza; Morgan Kaufmann, 1999.

[Saygin et al 2000] Turing Test: 50 Years Later. Ayse Pinar Saygin, Ilyas Cicekli, and Varol Akman; in *Minds and Machines (*Vol.10 No.4), pp 463-518.

[Schaeffer 1997] *One Jump Ahead*. Jonathan Schaeffer; Springer-Verlag, 1997.

[Turing 1950] Computing Machinery and Intelligence. Alan M Turing; in *Mind* (Vol.59 No.236), pp 433-460.

[USCF 2002] *The United States Chess Federation Rating System*. Mark Glickman, 2001; viewed at http://math.bu.edu/people/mg/ratings/rs/rs2.html on 14 March 2002.

Genetic Algorithms and Fine-Grained Topologies for Optimization

Xiaotong Wang
NuTech Solutions, Inc.,
28 Green Street,
Newbury, MA, 01951

Lawrence Davis
NuTech Solutions, Inc.,
28 Green Street,
Newbury, MA, 01951

Chunsheng Fu
NuTech Solutions, Inc.,
28 Green Street,
Newbury, MA, 01951

Abstract

In this paper we show how the performance of two meta-heuristic algorithms and two simple search routines varies as these algorithms are applied singly, in pairwise combinations, and in larger, finer-grained combinations. The area of application is f6 and f17, two well-known optimization benchmark problems. Our conclusion is that when these algorithms are combined in complex ways, their performance is much better than when they are used alone or in pairs, and so there is strong evidence that the current approach to optimization followed by many current practitioners with, for instance, an evolutionary algorithm succeeded by a hill-climber, could be improved on if more complex algorithm topologies were used.

1 MOTIVATION

This paper is designed to suggest that the approach currently used by many persons doing real-world optimization and doing optimization of test functions can be improved if those persons consider using combinations of optimization algorithms, rather than individual algorithms, or individual algorithms followed by a round of hill-climbing.

The conclusions of this paper will not be surprising to many readers, as they have been touched on before by other writers [e.g. Powell, Skolnick, and Tong 1991] each of whom has suggested that combinations of many algorithms can be noticeably more effective in optimization than those algorithms are in isolation or in pairs. The novelty in our paper stems from two sources. First, we will show that fine-grained algorithm topologies, in our parlance, do much better than simpler ones in optimizing. Second, we will show that for f6 and f17 (the test problems on which the results in this paper were generated) what arises from the best topologies is a type of algorithm topology different in type from those that many of us are use when we carry out optimization.

2 THE INITIAL PROBLEM

For our experiments, we used the test function f6, first suggested as an evolutionary computation benchmark by David Schaffer [1989]. This problem was used in extensive testing of evolutionary algorithms by Schaffer and his collaborators, and it was the problem that carried the weight of the expositional burden in the extended tutorial in [Davis, 1990]. It is a two-dimensional, damped sine wave that has been shown to be difficult for some types of hill-climbers and for simulated annealers. A cross-section of the f6 function is shown in Figure 1. The function is to be maximized, and so the optimal point is located at the center of the curve. The full function is generated by rotating the curve about its center point, so that it appears as a series of concentric ripples in a pond, with the highest ripple located at the very center of the pond. If one is searching for the optimal point without prior knowledge of the shape of the curve, it can be very difficult to locate. The function is extremely hilly, and since the ripple containing the optimal point is the smallest ripple in the pond, an extremely small part of the whole search space is occupied by the hill containing the optimal point.

The f6 function is formally stated as follows:

$$0.5 - \frac{(\sin \sqrt{x^2 + y^2})^2 - 0.5}{(1.0 + 0.001(x^2 + y^2))^2}$$

A solution to the f6 problem consists of two real-valued numbers x and y, each in the range between –100 and +100. The evaluation of such a solution is the value that the f6 functions returns when those numbers are plugged in as x and y.

3 ALGORITHMS USED

We used four algorithms in the experiments reported here:

- A random search (RS) algorithm that generates x, y pairs with uniform probability from the variable range of –100 to +100. The random search algorithm preserves the best solution it has found so far, and

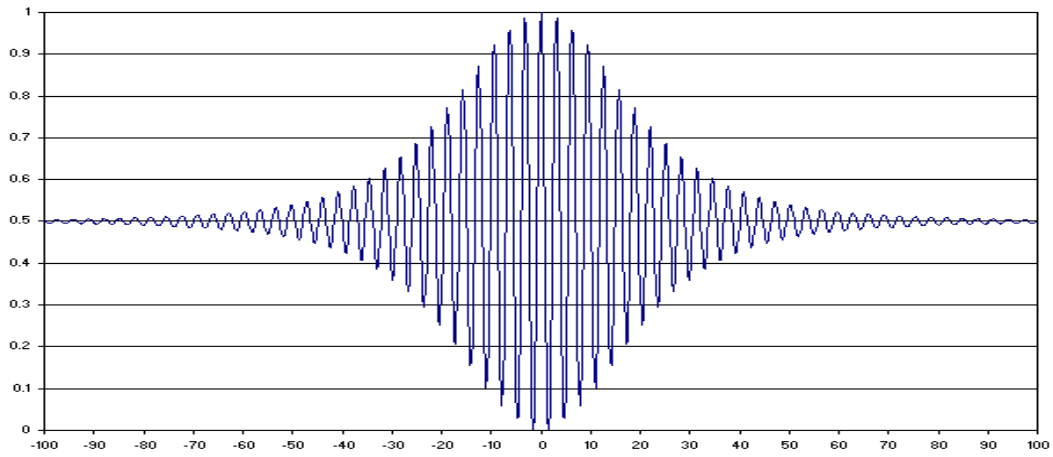


Figure 1. cross-section of f6

halts when it has carried out a predetermined number of evaluations.

- An opportunistic hill-climber (HC) that begins from a random position in the solution space. It maintains a current solution <x, y> and generates new solutions from a uniform distribution around the current x and y values. If the new solution is equal to or better than the current solution, then the new solution becomes the current solution. The hill-climber we use stops when it has carried out a predetermined number of evaluations.
- A simulated annealer (SA) that begins from a random position in the search space. The starting temperature is 2.0, and the cooling factor is 0.80. Our simulated annealer, given a number of evaluations n to work with, carries out n/20 of those evaluations at each temperature during optimization, and proceeds until it has carried out n evaluations.
- A genetic algorithm (GA) that begins with a randomly-generated set of solutions from the solution space. The genetic algorithm, given a number of evaluations n to work with, uses 10% of those as its population size, and proceeds until it has carried out n evaluations. The genetic algorithm is steady-state, allows no duplicates, represents solutions as lists of real values, and deletes the worst member of the population after a new solution is inserted.

4 INDIVIDUAL ALGORITHM PERFORMANCE

Many optimization systems, both commercial and academic, use a single algorithm or heuristic when optimizing. In cases when it is known that one will find the global optimum in the time available—when doing linear programming, for example, on a small problem—there is no need to consider alternatives, as long as one has sufficient computer resources with which to optimize. It is cases in which one is searching for good answers with limited computer resources that concern us here. What we will show below is that using any of our four algorithms in isolation to solve f6 and f17, or even using one followed by another for post-processing, is a strategy that is inferior to using fine-grained combinations of these algorithms.

Algorithm	Mean	Maximum evaluation	Minimum evaluation
HC	0.991236	1	0.986304
GA	0.990170	1	0.972698
SA	0.960624	1	0.933464
RS	0.959234	1	0.903464

Table 1. Single algorithm performance

Algorithm	Mean	algorithm	Mean
GA→HC	0.990848	HC→GA	0.990689
GA→SA	0.987373	HC→SA	0.990459
GA→RS	0.986847	HC→RS	0.990658
SA→HC	0.990585	RS→HC	0.990434
SA→GA	0.988114	RS→GA	0.988383
SA→RS	0.960609	RS→SA	0.962413

Table 2. Mean score of pairwise combination of algorithms

As Table 1 shows, our four algorithms vary widely in performance as they solve f6. Table 1 shows the mean, best, and worst scores for each of our algorithms after 3000 evaluations, as each algorithm solves f6 from a random start. The results in the table describe 500 runs of each algorithm, with a different random seed for each run. Table 1 shows us that, when using the same step size, the best single algorithm of these four for solving f6 is the hill-climber.

Let us now consider what the best pairwise combination of algorithms is. Table 2 shows the average of the best solution found for 500 runs each of each of the different seeding possibilities between our four algorithms. In each single run, one of the algorithms was run for 1500 evaluations and its best solution found was used as the seed for the second algorithm. Table 2 shows us that the best combination consists of a genetic algorithm seeding the hill-climber. The table also shows us that the performance of this combination is inferior to the performance of the hill-climber used alone. We do not show the maximum evaluation in this table since it is generally 1. The mean is the most significant value, while the minimum is less significant, so, we present the mean as the best representation of related algorithm pair performance.

It is possible that a different ratio of evaluations between these two algorithms would yield better solutions. However, this is not the case. Figure 2 shows the mean of 10 runs for each of the weight ratios between the hill-climber and the genetic algorithm, carried out at intervals of .2 and with 3000 evaluations per run. When evaluations are allotted to the genetic algorithm, performance is degraded. As the ratio rises to 3 or higher, the hill-climber is effectively carrying out search on its own, and the results are not significantly different for higher ratios.

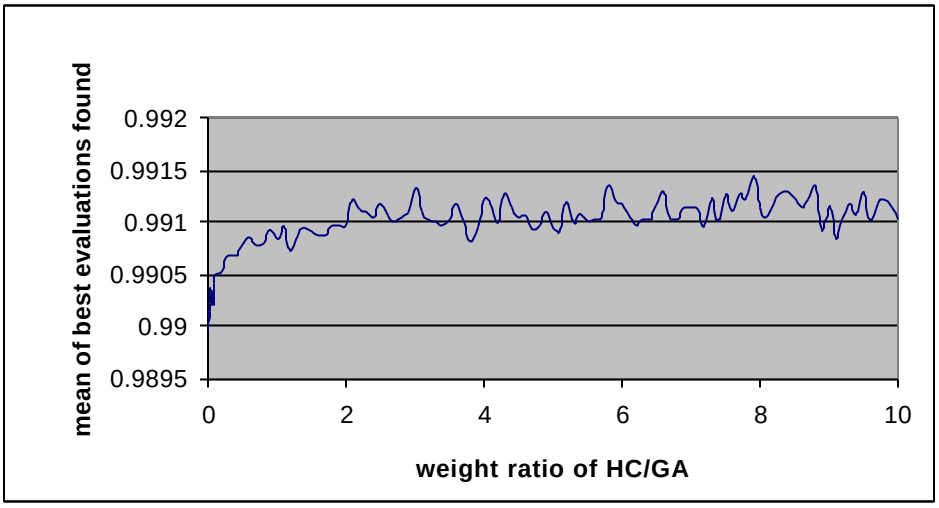


Figure 2. impact of weight ratios for F6

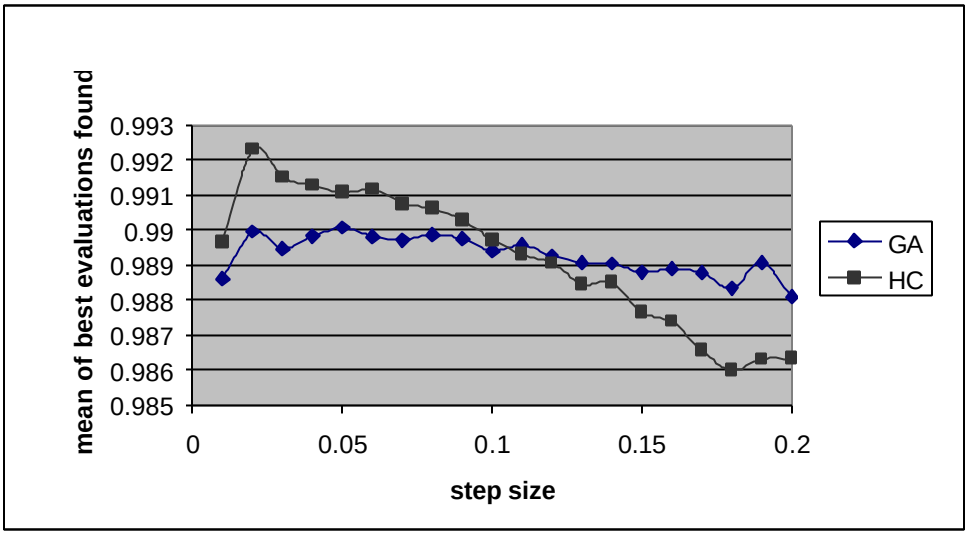


Figure 3. impact of step size for single GA and HC on f6

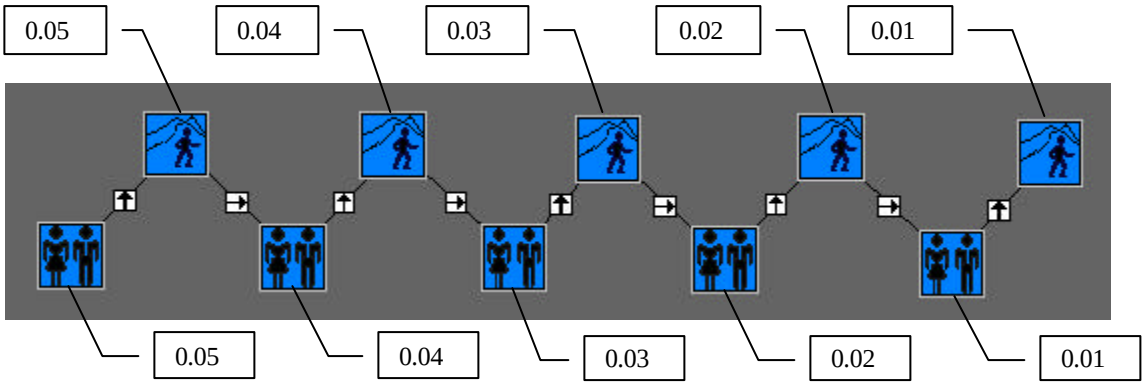


Figure 4. Step size distribution for a 5 GA → HC chain

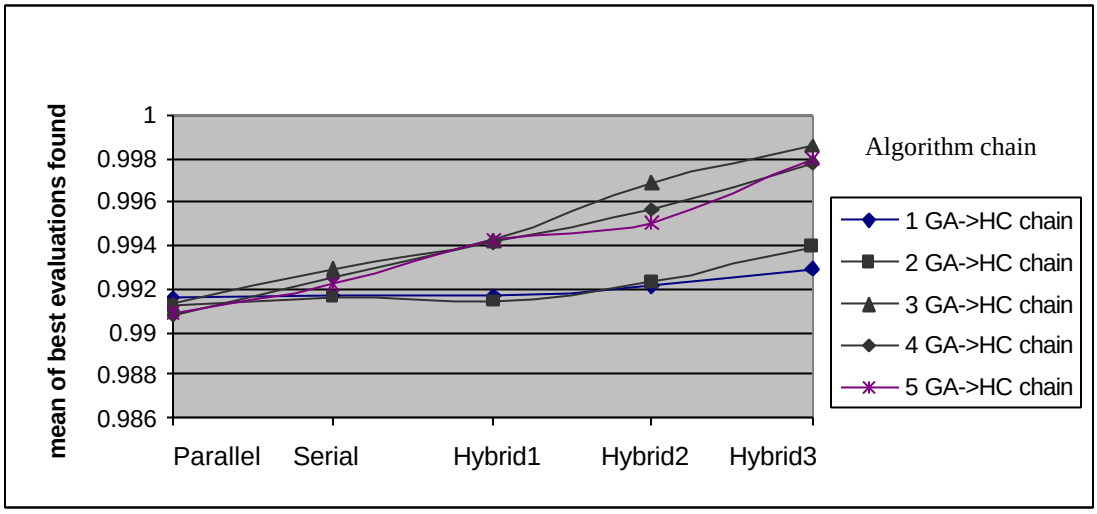


Figure 5. impact of number of algorithms and their configuration

5 THE IMPACT OF STEP SIZE

For the single-algorithm case and the two-algorithm case, the best algorithm is the hill-climber—especially interesting, when we remember that f6 is an algorithm that looks like it should frustrate hill-climbers. Why is it that the hill-climber does so well on a problem with a large number of hills? The answer is that the maximum step size used by the hill-climber in these experiments is .05, which allows the algorithm to jump from peak to peak in the search space. We set the step size at .05 for the three algorithms that use step size, having found empirically that it gave the best collective performance. Perhaps settings tailored for each algorithm will give better individual performance.

Figure 3 shows the effects of varying step size for the genetic algorithm and for hill-climber, considering their performance alone. The optimal step size for the hill-climber, considered in isolation, is actually about .02, whereas for the genetic algorithm the optimal value has less impact on performance, but lies at about .07. We see from Figure 2 that when the step size is set at .02, the hill-climber does even better than in the experiments described above, showing mean performance above .992 at that setting.

6 MORE COMPLICATED ALGORITHM TOPOLOGIES

In the remainder of this paper, we shall consider more complicated algorithm topologies. We use the term "algorithm topology" to refer to a set of algorithms linked to form a directed, acyclic graph whose links represent seeding relationships. These algorithms may be of the same type or of different types, and individual members of the topology may have parameter settings that vary from other algorithms of the same type in the topology. For example, Figure 4 shows a topology consisting of five hill-climbers (represented by an icon containing a hill-climbing figure) and five genetic algorithms (represented by an icon containing a male-female couple). This topology has seeding relationships that proceed from genetic algorithm to hill-climber to genetic algorithm, and so forth. The different hill-climbers and genetic algorithms have different settings for the step size parameter. The first algorithm to run in the topology will be the leftmost genetic algorithm, which has a step size of .05. Its best solution will seed the leftmost hill-climber, which also has a step size of .05. The hill-climber's best solution seeds the second genetic algorithm, which has a step size of .04, and so forth. The effect of running this topology will be that we will alternate between an evolutionary approach and a hill-climbing approach, always using the best solution found as a seed for the next algorithm. As we proceed through the run, the step size of the mutation operator will decrease in size until, on the final run of the genetic algorithm and the hill-climber, the step size is .01.

Figure 5 shows the results of running a number of different algorithm topologies on f6. In order to explain the results, we need first to explain what these algorithm topologies were.

Our experiments used algorithm topologies constructed from basic topology units. We used five different types of topology units in these experiments, and each was a variation on a basic GA→HC chain. In our notation, "1 GA→HC chain" refers to a single GA→HC pair, with the genetic algorithm seeding the hill-climber. The notation "2 GA→HC chain" describes two pairs of algorithms, each pair consisting of a genetic algorithm seeding a hill-climber. If this pair is run in serial fashion (Figure 6b), the output of the hill-climber in the first pair will seed the initial genetic algorithm of the second pair. If it is run in parallel (Figure 6a), then both pairs of algorithms will run without communicating, and the best solution found by either one will be returned as the solution found by the topology. To illustrate this, please refer again to Figure 4, which shows a 5 GA→HC chain run in serial fashion. As shown in Figure 4, we interpolated the step size for all algorithms in the topologies in the current set of experiments from .05 to .01.

In addition to experimenting with parallel and serial topologies of the GA→HC chains, we also experimented with higher-level configurations of the basic serial topologies. Figures 6c, 6d, and 6e show three types of hybrid topologies whose performance is reported in Figure 5.

Each of these three hybrids is a serial topology based upon the same components. Hybrid1, as shown in Figure 6c, is a serial topology with two similar component blocks, each of which, in this example, is a 3 GA→HC chain running serially. (Note that the output of *each* hill-climber in the first block seeds the genetic algorithm that begins the second block.) Hybrid2, as shown in Figure 6d, is a serial topology with three similar component blocks, connected serially. Hybrid3, as shown in Figure 6e, is a topology containing four similar component blocks, connected serially.

It should be noted that in our experiments, each of these topologies was given the same number of evaluations to use in solving the problem. The number of evaluations in these experiments is 3,000. In the case of a 1 GA → HC chain, performance with 3,000 evaluations was shown in Table 2. In the current set of experiments, the genetic algorithm runs for 1500 evaluations and seeds the hill-climber, which also runs for 1500 evaluations, and returns its best solution. Thus, for this configuration, we have 2 algorithms sharing the 3000 evaluations equally.

It is worth noting that for the 5 GA→HC chain run under the hybrid3 regime, we have 10 algorithms in each component times four components = 40 algorithms running. Since they share the same number of evaluations as the 1 GA→HC chain running serially, in this case each algorithm will have only 75 evaluations.

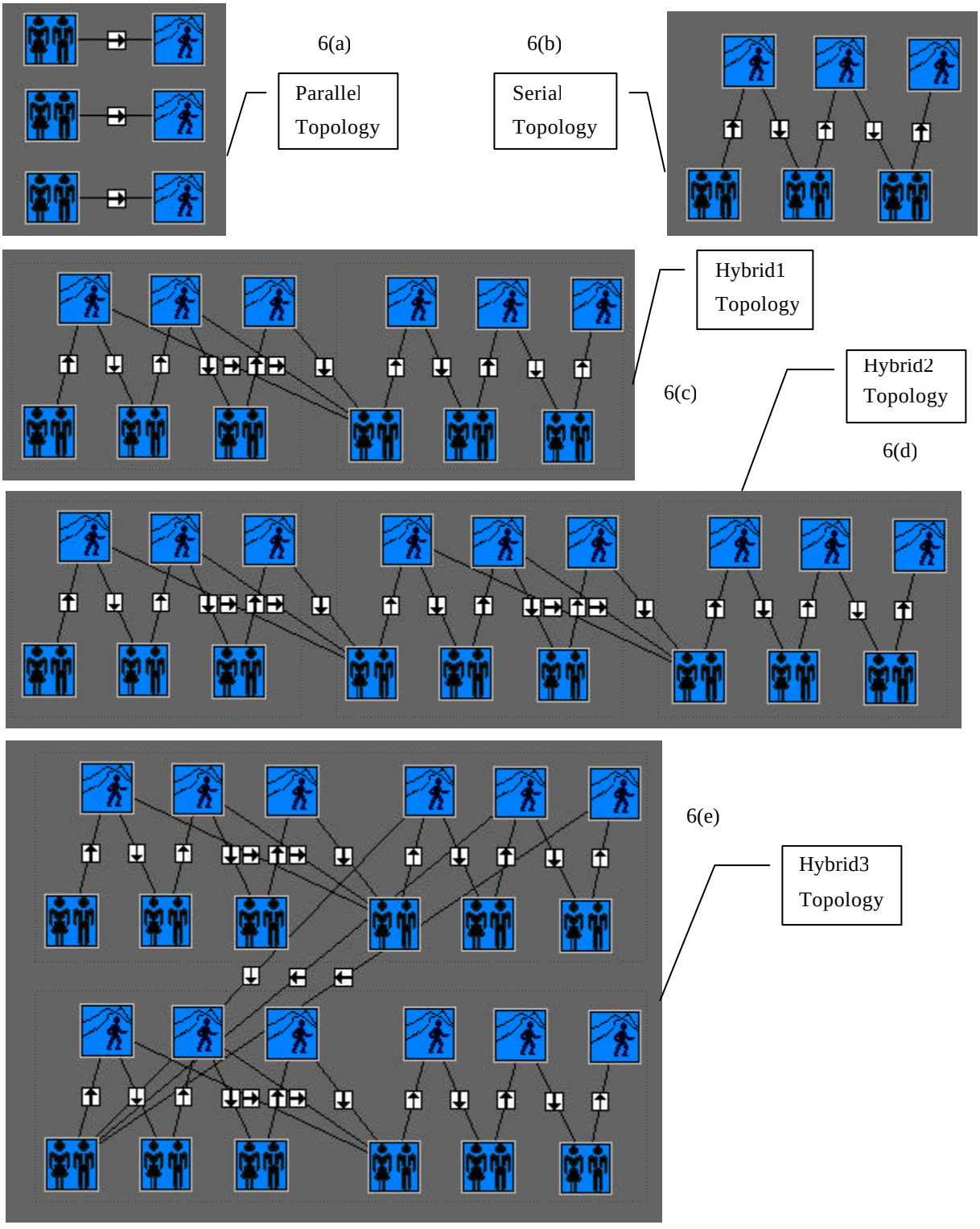


Figure 6. Topologies studied in Figure 5 using 3 GA → HC chains

The purpose of our experiments was to determine whether fine-grained topologies consisting of genetic algorithms and hill-climbers could do better than genetic algorithms and hill-climbers alone and, if so, just how fine the grain should be to get the best performance. Figure 6 shows the results. We see that the worst results were obtained by the parallel topologies, which consisted of independent GA→HC runs. The next-best results were obtained by the serial topology, which linked up those runs so that each GA→HC pair was seeded by the output of the prior pair. The next-best results were obtained by the hybrid1 topology, which duplicated the topology serial and seeded the second block of serial runs with the output of each hill-climber in the first block. The next-best was the hybrid2 topology, which used two serial blocks in serial. Finally, the hybrid3 topology used three serial blocks in serial, and did better than all other higher-level topologies.

It should be noted that each basic configuration—whether one, two, three, four, or five GA→HC pairs—did better under finer-grained topologies. It should also be noted that the best number of GA→HC pairs in the basic unit was 3. When four or five GA→HC pairs were used, results were not as good.

7 RESULTS FOR F17

In order to broaden the applicability of our results, we carried out similar studies for f17, a 30-dimensional, complicated mathematical optimization function described in (Baeck 1998). The results were similar. Using single algorithms, with parameters tuned to solve the problem, or using two algorithms in a seeding relation, optimal solutions could not be found by the best topologies in 10,000,000 evaluations. The convergence speed was so slow that we estimated it would take about 100,000,000,000 evaluations to get the optimal solutions. When we used the topology shown in Figure 7, with multiple iterations in which each run through the topology was seeded by the best result from the prior topology's run, the optimal solution was found, on average, in about 1,750,000 evaluations.

8 CONCLUSIONS

We have two principal conclusions from the results presented here. First, when the number of evaluations allowed is the same, the performance of simple topologies of our four types of optimization algorithms is vastly inferior to fine-grained, serial topologies of those algorithms. Second, we have shown that modifying the parameter values of those topologies across the optimization process leads to better results than holding them constant. We have shown that these conclusions obtain for the classical f6 function with two real-valued inputs, and for the much more difficult f17 function with 30 real-valued inputs.

Given that most current practice in evolutionary computation involves the use of simple topologies, we hope that these results will suggest refinements to our current practice that will allow us to find better solutions in the same amount of time.

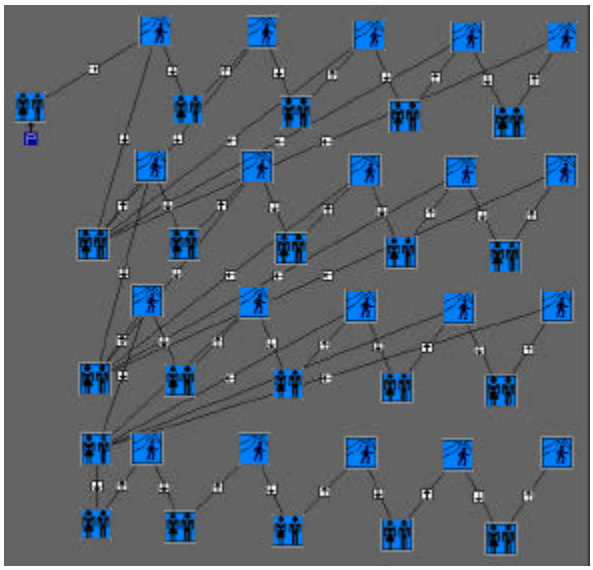


Figure 7 A best topology for f17

Note and Acknowledgement

We used HEURO (a system built by NuTech Solutions, Inc.) to display our topologies and gather data on our experimental runs. HEURO is a tool that makes it easy to create, edit, display, and evaluate the performance of algorithm topologies.

The authors would also like to thank Thomas Baeck for sharing his implementation of f17 with us.

References

T. Baeck and B. Naujoks (1998). Innovative methodologies in evolution strategies. INGENET Project Report D 2.2, Center for Applied Systems Analysis (CASA), Informatik Centrum Dortmund.

L. Davis (1990). *Handbook of Genetic Algorithms*, International Thompson Computer Press.

J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In J. D. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms, pages 51--60. Fitness Morgan Kauffman.

D. Powell, M. Skolnick, and S. Tong (1990). EnGENEous: A Unified Approach to Design Optimization. *Applications of Artificial Intelligence in Engineering* V (edited by J.S. Gero), Computational Mechanics Publications.