Basic Genetic-Algorithm-Neural-Network (GANN) Pattern with a Self-Organizing Security Example

Paper No. ICCST-2012-42

David Streisand
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030
USA

Rick Dove
Member, IEEE
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030
USA

***Abstract*—**The anti-system adversarial community is characterized as a self-organizing system-of-systems, noted collectively for its leadership in rapid evolution and innovative advancement; widening the gap between security cost and security losses. It appears that system security strategy cannot hope to even achieve parity without a comparable self-organizing strategy. Toward that end a project is underway to catalog re-usable patterns of self-organizing security of many kinds, principally found in natural systems, but also seen in recent computational approaches. One class of pattern of special interest involves discovery of previously unseen threats and attacks. In general this class of pattern has aspects of learning, innovation, and evolution as capability objectives. The genetic algorithm is one such pattern. Another such pattern is seen in artificial neural networks. Combining the two into a Genetic Algorithm augmented Neural Network, often called GANN, has considerable recent history in the literature. Not many of these are directly related to security applications. Some security-application work shows GANNs employed for feature selection provide enhanced learning performance and accuracy, and avoidance of local minimum traps. This paper adds the GANN pattern to the self-organizing security pattern catalog, and applies the pattern to a self-organizing security application under development.

Index Terms—GANN, intrusion detection, SAREPH.

I. INTRODUCTION

This paper is part of an on-going open effort started in 2010 [1] to develop a pattern catalog and pattern language for self-organizing security strategies. The project is motivated by the observation that the adversarial community is leading the generally-reactive security community in both attack innovation and rapid evolution; accomplishing this with effective self-organizing behaviors. Six characteristics (described later) that underpin adversarial evolutionary and innovative behavior have been abstracted and shown in Table I, for use in qualifying candidate self-organizing security patterns that may level the playing field.

The effort at this stage is developing a preliminary set of candidate patterns, and using them to refine an established pattern description form and the set of pattern qualifying characteristics [2, 3]. Refinement has not yet found a need to alter the basic architecture of the pattern form or the qualifying characteristics – but is rather clarifying fine distinctions between the qualifying characteristics and refining the descriptive nature of the pattern form elements.

TABLE I
PATTERN QUALIFICATION SAREPH FILTERS

[S]	**Self-organizing** – with humans embedded in the loop, or with systemic mechanisms.
[A]	**Adapting** to unpredictable situations – with reconfigurable, readily employed resources.
[R]	**Reactively resilient** – able to continue, perhaps with reduced functionality, while recovering.
[E]	**Evolving** with a changing environment – driven by situation and fitness evaluation.
[P]	**Proactively innovative** – acting preemptively, perhaps unpredictably, to gain advantage.
[H]	**Harmonious** with system purpose – aiding rather than degrading system/user productivity.

Neither the pattern project nor this paper proposes new patterns, but rather finds patterns in repetitive use that can be abstracted and described in a manner that facilitates understanding of the essence of the pattern, benefits conveyed by employment of the pattern, and a context in which it is appropriate for employment consideration. The descriptive method attempts to be comfortably understandable to systems engineers, security engineers, and decision makers involved in considering pattern employment.

The ultimate goal of the project is a pattern language for self-organizing security strategies. Using spoken language as a metaphor, identifying candidate patterns is akin to developing a vocabulary, and structuring a pattern language is akin to developing a grammar for combining patterns syntactically into semantically meaningful sentences [4].

This paper is the second foray into pattern combination, and here combines a basic genetic algorithm pattern [5] with an artificial neural network (ANN) as a candidate combination pattern (sentence, or perhaps sentence fragment in the pattern language). A basic ANN pattern has not yet been published as a stand-alone pattern, but shall be as a follow-on to this paper, which makes its nature obvious. The first combination of patterns employed the Bow Tie [2] pattern within the Proactive Anomaly Search [3] pattern.

In the literature the combination of a GA with an ANN is referred to as a GANN (genetic algorithm – neural network).

GANNs are applicable when it is necessary to learn or discover globally optimal solutions in complex situations and environments, beyond human capability for discovery and mathematical reduction. GANNs evolve innovative responses to cope effectively in complex and changing environments, and are likely to be an important strategy for dealing with zero-day attacks and advanced persistent threats – where prior attack knowledge is lacking.

The purpose for the pattern project was reviewed above. Next, the general nature of an ANN and a GA is introduced. Then an application example of GANN employment is reviewed and used to abstract the elements of a Basic GANN pattern. The GANN pattern is then applied to a self-organizing security example under current development.

II. ANN INTRODUCTION

An ANN is a computational construct, inspired by the parallelism of biological nervous systems, that can often discover and resolve complex relationships between many inputs in combination, to arrive at an optimal output for a specific and bounded problem space. An ANN is generally useful when the complexity of the relationships between input values and optimal outputs is beyond human ability to define – thus an ANN is used to learn the optimal relationships in a trial and error training/learning series that converges.

Briefly, and simplifying: ANNs consist of nodes (often called neurons) and node connections (often called synapses). Nodes are of three types: input layer, output layer, and intermediate (hidden layer) nodes. Connections between nodes may be feedforward only, or may include some looping feedback connections in a multi-layer ANN. ANNs have synapse connections between neurons that weight the value sent from a connected neuron. Weights may take on positive, negative, or zero values, and may be integer or real value depending on the application. In either case a weight value of zero is effectively a null connection and a weight value of 1 is a direct pass-on. The neuron is a transform function that passes on the sum of the weighted values of all of its inputs as its output, and may also apply a bias, as shown in Fig. 1. Inputs to a neuron are typically numerical values, or strings of features composing a pattern.

Topology of an ANN is expressed in the structure of nodes and connections, which may be fixed or evolving. A simple evolution implementation can be accomplished by alternating connection weights between zero value and some value. More complex topology evolutions will also change the number and location of nodes. Feedforward ANNs adjust node output values typically in waves, a wave moving synchronously through the ANN one layer of nodes at a time. A simple feedforward ANN has only input and output nodes, and calculates the output value in a single wave. A still basic ANN has feedforward connections through multiple layers of nodes, and completes a cycle when the calculation waves reach the final output nodes. An ANN with feedback loops is more expensive in computation, may take a while to settle a stable output set, and may also require some special settling computations to avoid infinite re-calculation loops.

An ANN produces outputs based on a fixed set of learned or trained neuron parameters. If it is a dynamic ANN it will have an internal ability to adjust these parameters in continuous incremental learning, generally driven by an external source of feedback that provides an error measure of fitness that is back propagated through the ANN layers. There are other forms of feedback learning as well.

A very basic ANN is feedforward only, with fixed parameters in one set of weighted synapses between inputs and outputs. A more generalized version of a basic ANN remains feedforward only, with additional "hidden" neuron layers between inputs and outputs.

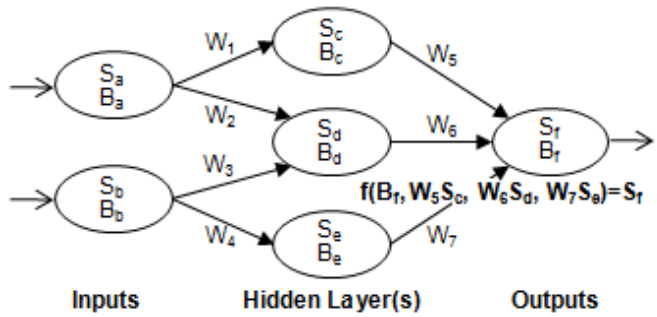


Fig. 1 – Components and transfer function of a single-hidden-layer feedforward artificial neural network (ANN)

III. GA INTRODUCTION

Briefly and simplifying: a genetic algorithm converges on a solution to an optimization problem by using a "fitness function" to evaluate how well candidate solutions address the problem. In natural evolution, fitness is the organism's ability to survive and reproduce. Computing applications, on the other hand, abstract fitness to match the problem at hand. In robotics, for example, fitness may represent a robot's ability to navigate around obstacles successfully. In security, fitness may represent the ability to detect never-seen-before intrusions with low false-positive and false-negative rates.

In keeping with the evolutionary metaphor, embodied candidate solutions may be called "individuals," the encoding of each candidate solution is called a "chromosome" or "genotype", and each distinct element within a chromosome is called a "gene".

With a chromosome structure established, the GA applies "operators", based on natural evolution, to evolve an initial randomly-generated (un-fit) population of individuals into more fit individuals, until a suitably fit individual emerges. The GA creates new generations of individuals from previous generations, evolving their chromosomes until a time limit is reached or until sufficient fitness is achieved.

To create a new generation of chromosomes, the fitness of the individuals in the current generation is evaluated using the fitness function. The GA then selects the fittest individuals stochastically, i.e., with some randomization, for carryover and modification for the next generation . The fittest individuals are selected for carryover as well as for breeding new individuals for the next generation with the GA *crossover* operator, Crossover mimics sexual reproduction in nature – two individuals are selected to breed, and their children's chromosomes are created by swapping a portion from each parents' chromosome at a randomly-selected location. Some other moderately fit individuals in the current generation are allowed to survive to the next generation, and the least fit individuals are replaced by the new products of crossover to

maintain the overall population size. The other commonly applied GA operator is *mutation* – occasionally, a random gene (bit or string of bits) of a chromosome is changed for the next generation.

IV. BASIC GANN PATTERN

A self-training ANN can be computationally expensive. The excellent paper by Montana and Davis [6] provide an example of using a GA to optimize ANN parameters, noting: "It not only succeeds in its task but it outperforms back propagation, the standard training algorithm, on a difficult example. This success comes from tailoring the genetic algorithm to the domain of training neural networks." Floreano Dürr, and Mattiussi [7] adds good breadth and depth to understanding GANNS, with numerous applications cited.

Dewri [8] provided the basis for Fig. 2. with terminology modified for consistency here. Dewri notes in his discourse on evolutionary algorithms (EA): "The evolution of connection weights introduces an adaptive and global approach to training. Unlike *gradient-based* training methods, viz. back propagation, EAs [GAs included] rely on probabilistic search techniques and so, even though their search space is bigger, they can ensure that better solutions are being generated over generations. Optimal network architectures can also be evolved to fit a given task at hand."

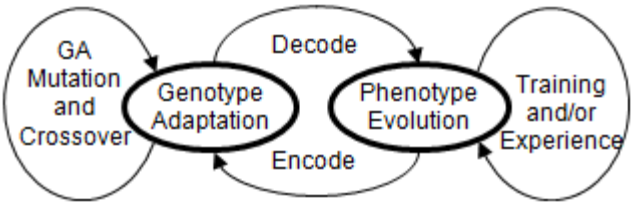


Fig. 2 – ANN network evolution with GA learning assistance (modified graphic from [8])

The GANN pattern is applicable to many problem domains, including security related applications. Some examples are cited in the bottom of the pattern form shown in Table III.

Here the model for the *basic* GANN pattern uses the evolution of predator-prey behaviors from Floreano and Keller [9] as a grounded example, combining a feedforward ANN through potentially multiple layers, with the basic GA pattern that employs crossover and mutation operators as described in [10].

Floreano and Keller provide many examples of complex behaviors emerging from GANN employment in robot mobility controllers: collision avoidance, cooperative behavior, prey stalking, and predator avoidance. Here the focus is on their predator-prey experiments. A population of predator robots and a population of prey robots were constructed on the same basic hardware platform, with the same neural network controller software, but differing in the physical limitations of their top maximum speed and vision capabilities. The prey could outrun the predator, but the predator had superior detection range.

Self-Organization – The GA for each robot self-organizes new chromosomes for robots that do not exhibit sufficiently fit behavior, using crossover and mutation GA operators. The fitness of robotic behavior is determined by the effectiveness of the ANN in successive generations of predator-prey co-evolution. The GAs also reorganize the population of ANN behaviors by loading new chromosomes into ANN connection weight parameters. Feedforward ANNs, as used in the predator-prey example, are not by themselves self organizing, and rely on the GA to provide the self-organization of the ANN structure.

Adapting to unpredictable situations – Adaptation occurs within the GA when it reconfigures chromosomes with crossover and mutation operators, working on stochastically selected chromosomes from among those with high and promising fitness. This adaptation can accommodate changes in the environment (performance of opponents). As the prey became more adept at avoiding predation, the predators adapt to their own less-fit condition and cause new behavior strategies to evolve, which in turn caused the co-evolutionary cycle. In all cases the behavior-evaluation of the ANNs established the fitness measure for the GAs.

Reactively Resilient – The GA generates new chromosomes with crossover and mutation operators rather than wholesale random replacement – using redundant gene fragments and diversity among gene fragments to enable recovery from a degraded average fitness rather than a catastrophic failure of the population . This has the effect of maintaining some prior history in new chromosomes, which lends resilience when chromosome changes are less fit in a new generation than in a prior generation. The GA feeds the ANN with new chromosomes to try. In combination the GANN exhibits resilient behavior through generations – recovering from setbacks as the environment changes.

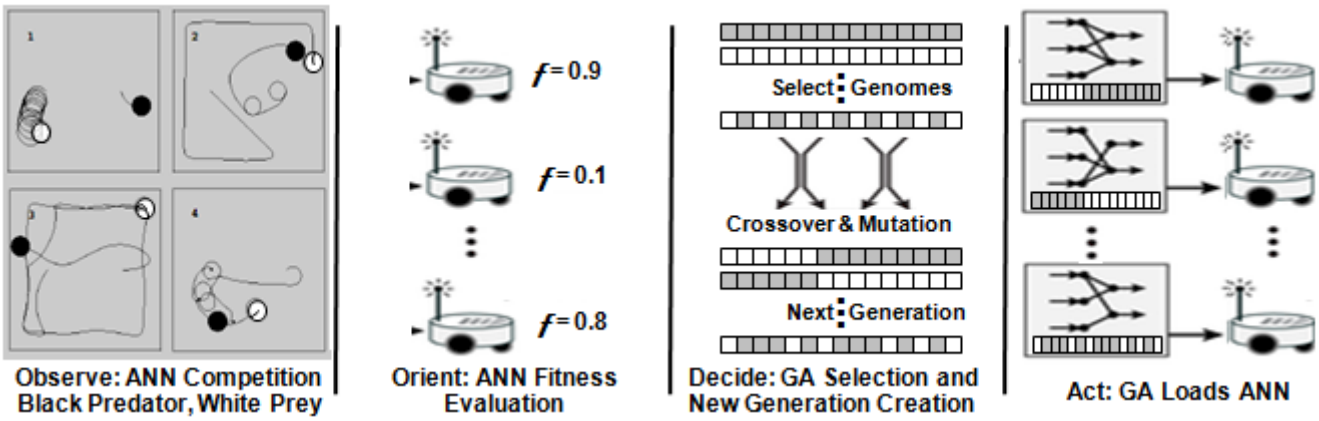
Evolvable Strategies – Evolution occurs across generations, is directed toward more fitness as opposed to random change, and relies on memory of prior generation genetic fragments and chromosomes to move average population fitness in the optimal direction. The predator-prey GANN example demonstrate an evolution of strategies that are generated and selected to take advantage of an adversary's weaknesses, while exploiting the protagonist's strengths. Initial populations of both types of robots had members wandering at random and at varying speeds. Under selective pressure, members of the prey population who failed to use their speed advantage to avoid losing a pursuit by a predator died off and were replaced; while the predator population developed members who used sophisticated visual tracking and motion control behaviors to intercept prey from outside the prey's sensory field of view. In time, the predators became so adept at catching prey in the open, that they lost their wall avoidance behavior. As the prey evolved to become harder to catch, the predators re-evolved their wall avoidance behavior. After several generations of being hunted by predators who largely avoided walls, the prey evolved a strategy of escaping by zooming along the walls. This induced a predator counter strategy of backing up against a wall, and waiting for the relatively blind prey to run into a predator.

Proactively innovative – A properly implemented GA is by nature proactively innovative; it creates speculative chromosomes in a search of superior chromosomes, and if permitted by the system designer, will continue to do so even after acceptable chromosomes are found. GA's that continue to search can avoid local and temporary optima. In the predator-prey example of co-evolution, the GANN reacts to a clearly unfit chromosome (behavior strategy), but is proactive

in the way the GA searches for solutions as well as in the GANN's ability to continue searching for better solutions beyond acceptable – design goals permitting. If a new chromosome is neither advantageous nor disadvantageous, it may persist in a percentage of a population for some time. If the fitness landscape shifts to confer a selective advantage on the new genotype, this can also be viewed as proactive innovation, held in reserve until useful.

Harmonious with system purpose – A properly designed GANN is inherently in harmony with the needs of its parent system, robot population survival in this example. Harmony between generations of predator or prey is preserved to the extent that GA operators of mutation and crossover evolve the incrementally toward more fitness.

TABLE III
BASIC GANN PATTERN

Name - Basic GANN
Context – A complex relationship, perhaps dynamic, between sensed environmental input and appropriate environmental response, beyond human capability to reduce to an algorithm, but appropriate for an evolutionary (self learning) algorithm.
Problem – An evolutionary algorithm that needs to converge rapidly, avoid local optima, and perhaps deal with a changing environment.
Forces - Complexity of the environment vs. capacity (size) of the ANN; speed of learning a static environment vs. generalization for new environments; speed of learning vs. cost of learning; suitability of fitness function vs. cost of evaluation; capability of the ANN (for classification) vs. suitability of the GA (genome structure) [10].
Solution – Combining a genetic algorithm that employs crossover and mutation operators to feed speculative parameters to a feedforward (only) single or multilayer ANN.
 GANN continuous loop from panel 4 back to panel 1 for successive generations. 100 robots per generation. Predators and prey converge from initially-random neuron genes to best-strategy genes in 300 generations. Fitness determined in fixed time limit by how close a predator got to a prey. Modified graphics from [9, 11].
[S]elf organization – occurs as GAs reorganize genes in chromosomes, and as ANN loading reorganizes the nature of the ANNs in the population. [A]daptation – GAs adapt to fitness evaluations, applying crossover and mutation operators to reconfigure chromosomes at a varying rate determined by the average fitness of the ANN population, permitting adaptation to a changed environment. [R]eactive resilience – is enabled by redundancy and diversity in GA chromosome fragments, so that an environmental change (superior opponent capability evolution) only degrades but does not destroy average fitness of the ANN population. Recovery is then enabled by GA adaptation and ANN evolution. [E]volution – ANNs evolve toward an optimal fit with a stable environment, caused by the GAs selectively retaining (remembering) chromosomes and chromosome fragments that contribute to better than average fitness. [P]roactive innovation – Horizontal gene transfer, a prime source of innovation [2], occurs with the GA crossover operator, stochastically combining gene fragments from selected successful chromosomes for insertion in next generation chromosomes. Mutation of gene fragments also contributes, by retaining most of a successful gene and proactively experimenting with small chromosome changes. Stochastic retention of gene fragments and moderately-fit GAs also contributes, by generating chromosomes and gene fragments that may be highly-fit when the environment changes. Finally, continued GA adaptation after a population has sufficient fitness proactively searches for opportunistic fitness. [H]armonious operation – Predator and prey GANNs are in harmony with the survival needs of their respective populations, evolving toward higher average survival fitness without retaining population-threatening survival obstacles.
Example – Co-evolution of predator and prey behaviors in robots [9, 11] **Example –** Very informative in GANN generality, with a sonar example [6]. **Example –** Very informative in GANN generality, with examples cited from many domains [7]. **Example** – Network backdoor intrusion detection [12] **Example –** GANN short-term forecasting framework for database intrusion prediction system [13].

V. GANN SECURITY APPLICATION

Anomaly detection promises to find elements of abnormality in a field of data. High end applications include finding patterns in unstructured data, identifying emergent behaviors in multi-agent systems, illuminating insider threats in progress, and detecting cyber attack vectors unseen previously.

Anomalous behavior detection promises a way around the limitations of looking only for known attack patterns, but it has raised issues of higher false positive rates and questionable normal-behavior stability. The difficulty stems from current technology approaches that cannot cover the entire field of possible anomalies; making compromises on pattern specificity and pattern capacity. Described here and depicted in Fig. 3 is an architecture that doesn't suffer from those compromises.

The example provided here is an extension-in-process of prior work conducted under a DHS S&T contract completed in 2011, and described in Self-Organizing Resilient Network Sensing (SornS) with Very Large Scale Anomaly Detection [14].

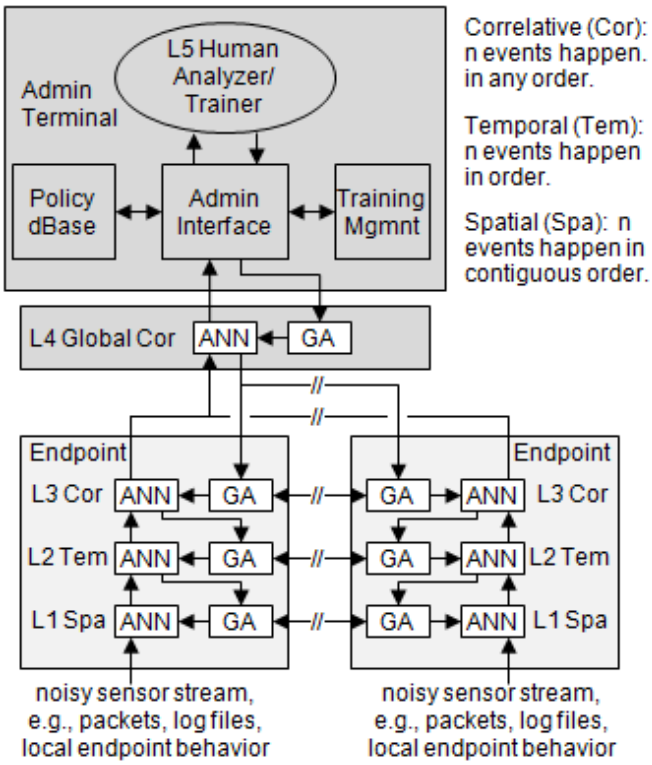


Fig. 3 – A 4-layer GANN. The GA at each layer receives fitness feedback from the next higher ANN layer, as guidance for evolving genes in same-layer ANN neurons. Feed-side collaboration among endpoint GAs improves learning speed at similar endpoints.

Above the GANN at L5, a human in diminishing supervised-leaning trains the GANN by evaluating what the GANN's fourth layer presents as suspicious, which causes primitive ANN layers to evolve into ones customized to the specific environment. A GANN layer at L4, similar to an L3 correlative endpoint GANN layer learns to correlate network wide anomalies and relieve the human of repetitive decisions and interventions. A human also retrains the GANN when changes in network resources, operational practices, or policy changes cause new lower layer anomalies. A human also trains lower level GANN layers to distinguish when anomalies do and do not need to trigger immediate automated action.

Contents of an ANN "neuron" are GA-provided "genes", composed of some maximum but small number n (perhaps 6 or so) 8-bit feature bytes received from the next lower level, where n will be established and fixed after additional experimentation. Genes may have trailing features set to null (don't care) when shorter genes are appropriate. If an ANN neuron has all non-null gene features satisfied, the neuron's output is a single feature passed to all neurons at the next ANN level.

Relevance (fitness) feedback propagates from the top layer down, helping a layer-specific GA determine what genes to keep and whet genes to replace at that level. The GA never stops generating some percentage of new genes, even when an optimal set appears to be present at the companion ANN layer, avoiding local minima traps and accommodating changing environmental conditions. This GANN relationship is bi-directional – the GA provides the ANN neuron content and the ANN provides the fitness function for the GA. It is expected that convergence on sparse (minimally-overlapping) ANN-layer genes will come from continued GA speculative gene generation, which should increase average fitness across an ANN layer as overlap is reduced.

The ANN neurons will be implemented in an inexpensive massively parallel VLSI chip, soon to market but currently being emulated, which will provide propagation through the ANN levels at input data-stream speed. Speed is important in the ANN as this is where learned attacks and suspicious behavior will be recognized immediately for recommended or automated action. Learning, driven by the GA, is not as time critical. Initially useful dictionaries at each level will take some time (unknown as yet) to converge, though seeding the ANN with known gene relevance can hasten initial convergence for a the local-application environment. In any event, the ANN can converge on useful content through in-actionable observation of real-time data before it is permitted to cause action with its conclusions.

The GA learning objective is to converge upon an optimal fixed-number of genes (the genetic dictionary) at each ANN level, where optimal fitness is a function of relevance to the next higher ANN level with minimal overlap at this level (sparse coverage within a very large range of potential genes). The GA loads each ANN neuron with a specific different gene.

Taking a lesson from reverse engineered and computer modeled mammalian visual cortex, it appears that very good "immediate" object classification occurs in the first four neural layers, with a dictionary of about 100 features in each of the first two layers. Subsequent-layer dictionary-size has not been seen in the literature by this author, but the regular repetitive nature of cortex leads a working assumption that they may be in the same size range.

As a benchmark, a four-layer GANN with 100 genes at each layer, would make 100 million classifications at the fourth layer. Human expertise in a given domain is thought to

require 200,000 to 1 million pattern classifications. Even if 99 percent of the GANN's top level classifications are non-sense, expertise appears possible.

The first layer of one portion of modeled visual cortex looks only for edges, composed of adjacent retinal pixels in various orientations and sizes, numbering approximately 100. The classified edges are then sent to the next higher layer that looks for certain (on the order of 100) adjacent edge configurations that are classified as (so-called) glyphs. Certain configurations of adjacent glyphs are classified at the next level as (so called) geons, with the fourth layer able to identify general object classes, such as a car, a face, a house, etc. It appears that cortex looks for anomalies at each of these levels – those configurations that are relatively rare but also useful for distinguishing relevance from the large field of possible input data. Here anomalies are not bad things, but rather good things which afford optimal incremental sense making.

Genes for GA replacement speculations are not exclusively (or necessarily) generated by mutation and crossover operators acting on parent (prior generation) genes, but rather chosen from a very large (10 to the power 15) dynamic catalog of known anomalous 6-feature candidates (not-normally-seen-recently gene configurations). This is enabled at full data stream speed by the VLSI pattern processor and a special anomalous-gene detection architecture referred to as a Gang Detector (GD) [14]. The GD is a massively parallel GA mutation operator, of a specialized nature, that provides pre-qualification on speculative gene candidates by weeding out often recently seen "normal" candidates. Crossover may have some use in temporal detector genes, but not in correlative detectors, and probably not in spatial detectors as the GD operator will explore the spatial configuration space more effectively. The GD is a GA that can establish anomaly-fitness of 10 to the power 15 genes simultaneously, and can cover the entire space of genetic possibility simultaneously at data-stream speed. GD creation and management is modeled on the immune system inspired Proactive Anomaly Search pattern [3] coupled with the BowTie pattern [2].

By themselves, the vast coverage of GDs can reduce false negatives, but not false positives, and may increase the occurrence of false positives due to greater coverage of anomalous pattern space. False positive reduction will be accomplished by two aspects of the overall architecture: (1) non-repetitive human evaluation and training at L5, and (2) human evaluation feedback will propagate as learning in lower ANN layers (memory) with companion immediate-action triggering at any level where appropriate.

The issue of normal-behavior stability in typical, but not all, cyber networks is addressed by at least two aspects of the overall architecture: (1) continuous re-generation of new GDs that will track normal behavior through changes in the environment; and (2) feedback directives from L5 that can temporarily suspend ANN memory changes and action triggers when temporary changes to the environment are made.

This GANN example is self-adaptive to local dynamics and provides custom anomaly detection with no two installations alike in learned pattern content, adding difficulty to an attack strategy that relies on pattern-detection knowledge across widespread monolithic installations.

The next phase of the SornS project is building and testing a VLSI-chip emulation of the example shown here, with an endpoint bump-on-the-wire prototype for network endpoints when the chip is available.

VI. CONCLUSION AND FURTHER WORK

A basic genetic algorithm was combined with a simple feedforward artificial neural network to produce a basic GANN pattern; described in the pattern-form (Table III) first in generic terms and then grounded with an illustrative predator-pray co-evolution example. Examples cited at the bottom of Table III provide additional instances of GANN applications in security as well as other domains. GANN pattern employment was then described in a self-organizing security architecture currently under development [14], as a way to make sense of anomalies in a vast "big-data" intrusion detection space, with low false positives and low false negatives.

The effort that developed the GANN pattern described in this paper followed immediately upon the work that developed the basic GA pattern [3], and as a result, contributed to a refinement of self-organizing pattern qualification description, evident in comparing [3] with this paper.

The SornS project [14] has and will employ a number of the patterns developed to date by the pattern project. GDs (described earlier) are modeled on the immune system inspired Proactive Anomaly Search pattern [3], coupled with the BowTie pattern [2], and are now in early stages of coupling in SornS with the Basic GANN pattern (this paper), the Basic Genetic Algorithm pattern [5], the Hierarchical Sense Making pattern [3], the Horizontal Meme Transfer pattern [2], and the Quorum Sensing pattern [15]; and it is anticipated that the Peer Behavior Monitoring pattern [1] and the Dynamic Phalanx Defense pattern will be subsequent additions.

Future pattern-project work anticipates developing the basic ANN pattern formally, adding advanced versions of GA and ANN patterns, and adding many more relevant patterns, such as fractal architectures, autocatalysis, and weak tied networks to name only a few. Another anticipated pattern-project effort will take evolved refinements in SAREPH descriptive approaches and update previously patterns for descriptive consistency.

VII. REFERENCES

[1] Rick Dove and Laura Shirey, "On Discovery and Display of Agile Security Patterns," Conference on System Engineering Research, Hoboken, NJ (US), 17-19 Mar. 2010. www.parshift.com/Files/PsiDocs/Pap100317Cser-OnDiscoveryAndDisplayOfAgileSecurityPatterns.pdf

[2] Rick Dove, "Pattern Qualifications and Examples of Next-Generation Agile System-Security Strategies," IEEE International Carnahan Conference on Security Technology (ICCST), San Jose, CA (US), 5-8 Oct. 2010. www.parshift.com/Files/PsiDocs/PatternQualificationsForAgileSecurity.pdf

[3] Rick Dove, "Patterns of Self-Organizing Agile Security for Resilient Network Situational Awareness and Sensemaking'" 8th International Conference on Information Technology: New Generations (ITNG), Las

Vegas, NV (US), 11-13 Apr, 2011. www.parshift.com/s/110411PatternsForSORNS.pdf
[4] Christopher Alexander, *A Pattern Language: Towns, Buildings, Constructio,* Oxford University Press, 1977.
[5] Rich Messenger and Rick Dove, "Basic Genetic Algorithm Pattern for Use In Self-Organizing Agile Security", IEEE International Carnahan Conference on Security Technology (ICCST), Boston, MA (US),15-18 Oct. 2012.
[6] David Montana and Lawrence Davis, "Training Feedforward Neural Networks Using Genetic Algorithms." In *International Joint Conference on Artificial Intelligence*, pp 762-767, Detroit, MI (US), 20-25 Aug. 1989.
[7] Dario Floreano, Peter Dürr, Claudio Mattiussi, Neuroevolution: from architectures to learning, Evolutionary Intelligence 1(1): 47–62, 2008.
[8] Rinku Dewri, Evolutionary Neural Networks: Design Methodologies, Published online by ai-depot, 24 Mar. 2003. http://ai-depot.com/articles/evolutionary-neural-networks-design-methodologies/
[9] Dario Floreano and Laurent Keller, "Evolution of Adaptive Behaviour in Robots by Means of Darwinian Selection," PLoS Biol 8(1): e1000292, Jan. 2010.
[10] Dong Ling Tong and Robert Mintram, "Genetic Algorithm-Neural Network (GANN): a study of neural network activation functions and depth of genetic algorithm search applied to feature selection," *International Journal of Machine Learning and Cybernetics*, 1:75–87, 2010.
[11] Dario Floreano and Stefano Nolfi, "God save the red queen! Competition in co-evolutionary robotics," In Koza, Kalyanmoy, Dorigo, Fogel, Garzon, Iba & Riolo (Eds.), Genetic Programming: Proceedings of the Second Annual Conference, pp. 398-406, San Francisco, CA (US), 13-16 Jul. 1997.
[12] E. Salimi and N. Arastouie, "Backdoor Detection System Using Artificial Neural Network and Genetic Algorithm." In *Procedings International Conference on Computational and Information Sciences (ICCIS) pp* 817-820. Chengdu, China, 21-23 Oct. 2011.
[13] P. Ramasubramanian , A. Kannan, "A genetic-algorithm based neural network short-term forecasting framework for database intrusion prediction system," Soft Computing 10(8): 699–714, May 2006.
[14] Rick Dove, "Self-Organizing Resilient Network Sensing (SornS) with Very Large Scale Anomaly Detection," Proceedings 2011 IEEE International Conference on Technologies for Homeland Security, Waltham, MA (US), 15-17 Nov. 2011. www.parshift.com/s/111115VeryLargeScaleAnomalyDetectionSorns.pdf.
[15] Jeff Hamar and Rick Dove, "A Quorum Sensing Pattern for Multi-Agent Self-Organizing Security Systems ", IEEE International Carnahan Conference on Security Technology (ICCST), Boston, MA (US),15-18 Oct. 2012.

VIII. VITA

David Streisand is a systems engineer in the U.S. defense industry, and during the past 33 years has been a software and systems engineer on defense and commercial systems, including unmanned aircraft systems, mission planning systems, and precision navigation and targeting systems. He holds a BS in Computer Science from California State University at Northridge, and is working on an ME in Systems Engineering from Stevens Institute of Technology.

Rick Dove develops agile self-organizing systems as a principle investigator and application program manager at Paradigm Shift International, and chairs the INCOSE working group for Systems Security Engineering. He is an adjunct professor at Stevens Institute of Technology in the School of Systems and Enterprises where he teaches basic and advanced courses in agile systems, and is author of *Response Ability – the Language, Structure, and Culture of the Agile Enterprise* (Wiley 2001). He co-led the OSD/Navy-funded project that identified systems agility as the new competitive frontier, and then led the research at the DARPA/NSF-funded Agility Forum. He holds a BSEE from Carnegie Mellon University, with graduate work in Computer Science at U.C. Berkeley.

IX. ACKNOWLEDGEMENTS

Some of the work covered in this paper was funded by the Department of Homeland Security under contract D10PC20039. The content of the material contained herein does not necessarily reflect the position or policy of the Government, and no official endorsement is implied.

James H. Burkhard, of Paradigm Shift International, built the SornS simulator and VLSI pattern-processor emulator , created test sets, and ran repeated tests to probe the operational characteristics of the Very Large Scale Anomaly Detector (gang detector).