Optimizing and Matching Bitstrings

JASON BROWNLEE
Technical Report 050604A
Complex Intelligent Systems Laboratory, Centre for Information Technology Research,
Faculty of Information and Communication Technologies, Swinburne University of Technology
Melbourne, Australia
jbrownlee@ict.swin.edu.au

***Abstract*-Strings of bits (bitstrings) are a common first-order representation in the design and preliminary investigation of computational intelligence algorithms given (1) the ease in mapping the strings to arbitrary domains (such as real numbers), and (2) in terms of mathematical analysis. This work considers the use of bitstrings in the context of the clonal expansion and antigenic selection in a general clonal selection algorithm, and provides a general review of mutation-based optimization (genetic algorithms) and bit-string matching (classifier systems and negative selection).**

Keywords- Binary, String, Bit-string, Matching, Optimization, Clonal Selection Algorithm

I. INTRODUCTION

A natural realisation of the clonal selection principle includes the choice of strings of bits as a first-order representation. The reason that this representation is a natural choice is because such a representation is (1) amenable to analysis and (2) amenable to remapping, and (3) provides a genetic metaphor facilitating genetic-like operations such as copying errors during cell division. The use of binary strings as a first-order representation is pervasive in computational intelligence, particularly in the field of evolutionary computation (genetic algorithms [4,21]). This work investigates the use of strings of bits (bitstrings) from two perspectives as they relate to the clonal selection principle and a natural realisation as a computational algorithm. The first perspective is of the maturation of clonal progeny via blind hypermutation (increased chance of copying error during cell proliferation). This will be investigated in the context of optimization algorithms that perform blind mutation on bitstrings. The second perspective is the selection of receptors by antigen modelled as the matching of strings of bits. This will be investigated in the context of algorithms that perform matching between bitstrings.

Section II provides a brief review of the optimization of bit strings with a focus on the genetic algorithm, including an analyses of the mutation operator, bitstring Hill Climber algorithms and test problems. Section III reviews algorithms that perform matching with bitstrings, with a focus on Learning Classifier Systems, and the Negative Selection Algorithm. Finally, section III provides some general comments on how the reviewed research may affect the design of general clonal selection algorithms.

II. OPTIMIZING BITSTRINGS WITH MUTATION

The genetic adaptive plan [21] which became the genetic algorithm [4] operates on a population of bitstrings. This representation facilitated the genetic operators of the approach, and the analysis of the behaviour of the system as processing building blocks (schemata theorem). One of the generalized genetic operators is mutation, the probabilistic flipping of bits in the bitstring during genetic replication. Holland proposed that mutation is determined *"by a random process where each position has a small probability of undergoing mutation, independently of what happens in other positions"* ([21], pg. 109). In using random mutation alone Holland suggested the use of small mutation probabilities such that the process provides a high probability of maintaining history, which is the dependence of the approach. Holland points out the use of high mutation values results in little dependence between successive generations of individuals, calling such an approach 'essentially enumerative'. Low probability mutation facilitates the dependence between observations and new trials, although is suggested to be 'very unsophisticated' compared to genetic crossover. He proposes the use of low probability mutation as an improvement operator that rather than to be used in the construction of new trials. The proposes the operator should be used to complement the primary genetic operator by adding back in lost genetic material. Further, the application of the mutation operator was described to be disrupting to the building blocks moved around by the crossover operator, further reinforcing the need for a low probability of occurrence.

Holland called mutation a background operator, and thus the effects of the operator were largely ignored, and the probability of point-mutations was conventionally kept very small, in the order of [0.001,0.01] [4,50]. Goldberg [7] differentiates between mutation and crossover by suggesting that *selection+mutation* provides continual improvement with limited scope, whereas *selection+crossover* provides innovation via jumping and remixing of components.

$$P[mutation] = 1 - (1 - p_m)^L$$

Equation 1 - Probability of a bitstring being mutated where P_m is the mutation rate and L is the string length

Eiben and Schippers [1] summarise the role of the mutation operator as a uninary (or asexual) and unbiased operator, highlighting its role as a main operator in

evolutionary strategies (traditionally real numbers) and the primary operator in evolution programming (traditionally operates on finite state machines).

Spears [51] highlights the complementary nature of the two primary genetic algorithm genetic operators suggesting that mutation is the focus if optimality is important and the crossover is useful accumulated payoff is important. He goes on to suggest that crossover and mutation are two versions of the same general operator that perturbs genetic representations based on available information. In his dissertation [52], Spears proposes that mutation has a high exploratory power (in that it is not dependent on the composition of the population) and no positional dependence (in that it has the potential to transform a given string into any other string in the search space).

Work by Schaffer, et al. in the late 1980's and early 1990's highlighted the untapped potential of the mutation operator, suggesting the operators power had been underestimated in genetic algorithms [16,17]. These works attempted to formalise an optimal mutation rate for genetic algorithms, work that was extended experimentally in [17] and theoretically [23]. This later work suggested the use of an initially large mutation rate that exponentially decreased with generational running time of the algorithm. A dynamic and decreasing mutation rate was further supported by others investigating the potential of the operator (not limited to [13,47]. The reason for this is "*[t]he probability that a mutation will give a better string decreases with the number of bits which are correct*" [49].

A. *The (1+1,m)-Algorithm*

A simplified genetic algorithm was proposed for investigating the strengths and limitations of the mutation operator, which became known as a *mutation hill-climber*, *(1+1,m)* [13] or simply *(1+1)* [49]. The algorithm has a static mutation probability (P_m) applied uniformly and independently to each bit in the bitstring. The algorithm was called a hill climber because without crossover, there is no inter-population interaction of genetic information, thus if a population is used, it is a collection of unrelated hill climbers. The (1+1) is taken from the Evolution Strategy naming convention ES($\mu+\lambda$) where μ parents create λ offspring, and the best from both '+' are selected. The scheme was reduced to a single parent and offspring for ease of analysis.

Step1: Generate a random string and accept as current string
Step2: Mutate each bit of the string with probability P_m
Step3: Accept the new string if its fitness is the same or better
Step4: If not stop condition, goto Step2

Figure 1 - Summary of the (1+1) algorithm

As a hill climbing algorithm, the approach has the interesting property that there is no well-defined neighbourhood for a given current string. Rather, a probabilistic neighbourhood is defined by the mutation probability. This neighbourhood covers the entire search space such that it is possible to move from a give string to any other string in the space, a probability that decreases with the increase in Hamming distance between strings and the current string. Muhlenbein [13] analysed the algorithm and proposed an approximate optimal mutation rate (the standard mutation rate), as follows:

$$P_m = \frac{1}{L}$$

Equation 2 - Approximation of the optimal mutation rate for (1+1)

Using the approximate optimal rate Equation 2, and the probability of a string being mutated Equation 1, a table of common bitstring lengths may be specified.

Length (L)	Approx. Optimal Mutation (P_m)	Prob. of 1-bit change to string
8	0.125	0.656
16	0.063	0.644
32	0.031	0.638
64	0.016	0.635
128	0.008	0.634
256	0.004	0.633

Table 1 - Example of heuristic mutation rate and chance of a mutated string for common bitstring lengths

Droste, et al. [43] point out that enumeration of all bits requires $O(2^L)$ evaluations, where as the (1+1) algorithm is guaranteed to find the optimum in at most $O(L^L)$ evaluations (which is obviously worse than enumeration). Muhlenbein [13] proves that the algorithm can solve the OneMax problem in $O(L\ lnL)$ evaluations (using the approximate optimal rate), and may apply to any linear binary function [43]. Rudolph proves [11] that the algorithm can solve LeadingOnes in $O(L^2)$, and Longpath in $O(L^3)$ [12]. Finally Droste, et al. [42,43] point out the importance of the algorithm to accept mutated strings with the same fitness (as well as improved) such that the algorithm may cross flat spots in the search domain.

The (1+1,m) algorithm may be extended to more than one parent and children, and beyond for example variable static mutation rates for children [40].

B. *Hill Climbers*

The (1+1) mutation hill-climbing algorithm is commonly compared to other bitstring based hill climbing algorithm (in addition to genetic algorithms and crossover-only algorithms). The following table provides a listing of the more common bitstring hill-climbing algorithms and a summary of their strategy.

Algorithm	Summary
Next Ascent Hill Climber (NAHC)	Mutate bits from left to right (ordered), evaluating strings as they are created. Accept the first best string as the current string (described in [45])
Random Bit-Climbing (RBC)	Create strings in a one-bit neighbourhood, evaluate in random order and accept the first best string ([27])
Steepest Ascent Hill Climber (SAHC)	Select the best fitness string from all points in the one-bit neighbourhood ([41])
Random Mutation Hill Climber (RMHC)	Select a random bit position and mutate, accept if improvement ([31,45])

Table 2 - Common bitstring hill climbers

C. *Test Problems*

There are many specially designed bitstring problems, specifically designed to highlight the limitations or advantages of the genetic algorithm, or

one of its operators such as crossover or mutation. These algorithms are typically benchmarked against the simple hill climbing methods already discussed. The functions are considered 'black-box' in that no special information about the environment is given other than the fitness scorings allocated to algorithm-proposed bitstring solutions.

OneMax and Related

One of the simplest bitstring functions is simply the sum of the number of 1's in the string. This function is called *OneMax*, (also called unitation or ones counting) [8]. As an objective function, it provides enough information to encourage improvement, without being too specific with regard to how to improve. Culberson [15,22] claims the function is easier for hill climbing than crossover as the crossover must in effect sort bits. An extension to *OneMax* is to sum the multiplication leading ones for each position. This is called the *LeadingOnes* function, proposed by Rudolph [11], and analysed by Droste, et al. [43]. Also proposed in the latter work investigating the (1+1) algorithm is a *Distance* function and a *Jump* function that provides a configurable gap between the optima and the leading trail.

Another simple extension proposed by Droste et al. [42] to simply return the product of all bits in the string. This produces a needle-in-a-haystack function called *Peak*, with a single optima when all bits are one. Muhlenbein [13] proposes another simple function called *Equal* in which the objective is to locate a string of equal number of ones and zeros. Back [49] proposes the optimization of a single integer value encoded using binary coded decimal or gray code in order to investigate the effects of such mappings on the search.

Royal Road

Mitchell, et al. [33,45] proposed the RoyalRoad function to investigate the crossover operators behaviour in the context of the building block hypothesis, and compare its performance to the RMHC [31,32]. The functions were designed to test what the crossover function is supposed to be good at – the processing and recruitment of building blocks. A number of variations of the function were proposed including the simple R1 function, which sum the number of fixed-position 8-bit building blocks contained in a 64-bit string, and the R2 function that includes the addition of 16-bit and 32-bit building blocks in the evaluation. Simple extensions include the $R2_{introns}$ (R3) function that extends R2 to include eight '*' (any bits) after each schema (doubling the problem in length to 128 bits), and $R2_{flat}$ where R2 is extended such that each of the 14 schema blocks have a flat scoring of 1 (linear), rather than the nonlinear increase in R2 (8, 16, 32).

Traps

Ackley [8] proposed a trap function which is investigated in the context of (1+1) by Droste, et al. in [43]. Trap functions are bitstring fitness functions that attempt to cause a search process to get stuck by sub-optimal solutions. Specifically, functions are designed such that sub-optimal solutions are easy to achieve and are a long distance from global optima, making it difficult for an algorithm like a hill climber to transition from a false to a true optima in a reasonable neighbourhood. Trap functions are also investigated in [40] in the context of the (1+1) algorithm and variants. Also see [25,30], and [34].

Fitness Distance Correlation (FDC)

Fitness distance correlation (FDC) is a candidate measure for predicting the difficulty for a genetic algorithm of achieving the global optimum of a binary problem [46,48]. Hamming distance is used between sets of bitstring and the optimum, taking into account the fitness of the strings. A random sample of bitstrings are taken from the problem, potted, and a correlation is measured. A large negative correlation implies the problem is easy to optimize for a genetic algorithm, large positive is difficult and deceptive for a genetic algorithm, and a near-zero correlation does not provide guidance toward or away from the optimum. Jones and Forrest [48] provide an excellent treatment of the state of binary optimization functions as they relate to each other with regard to the FDC. Altenberg provides a counterexample using a crossover-based distance measure [29]. The results are interesting and the measure has some support [37], as well as some concerns (for example with ridge functions [39]).

Deception

Goldberg proposed the construction of misleading or deceptive bitstring problems in order to investigate to boundaries of genetic algorithm (crossover operators) capabilities. Exploiting the building block hypothesis, he designed the minimally deceptive problem (MDP) such that fit low-order building blocks lead to a sub-optimal solutions [2], and [3] pg 46-52). A fully-deceptive 3-bit problem is proposed in [5]. Deception in 5-bit functions was proposed in [28] and in [10]. For an introduction to deceptive functions, see Whitley [26]. The work on deception was extended by concatenating six smaller bi-polar deceptive problems together to create a massively multimodal deceptive function [6].

Long Path (Root2path)

A final interesting class of problems proposed by Horn et al. [20] are called LongPath problems in that they provide a simple one-bit path to the global optimum that exponentially increase in length with the length of the bitstring (L). Such problems are thus intractable for one-bit hillclimbers as they are guaranteed to reach the optimum although it is impractical to wait for them to do so. Rudolph [12] provides a proof of a two-bit hill climber reaching the optima in polynomial time.

III. MATCHING BITSTRINGS

The matching of bit strings is common in artificial intelligence given the prolific use of bitstring's as first order representations. This section reviews the matching methods of two fields that are primary concerned with similarity between strings of bits: Learning Classifier Systems (LCS) and Negative Selection Algorithms (NSA).

A. *Learning Classifier System*

Learning classifier systems are a machine-learning paradigm that employ reinforcement-learning and

genetic algorithms and are capable of connectionist like information processing with a rule-system based representation [18,19] (see [38] for an overview of the field). Information in the system is represented as a population of classifiers, each with one or more conditions, and one action. Conditions are represented using a ternary alphabet [0, 1, #], where the '#' represents either a '1' or '0'. The action parts of the classifiers are traditional bitstrings. The classifiers match strings from an internal messages list, messages are binary strings posted by sensors (external input), or from the action part of triggered classifiers. A match is a binary decision (match or no match) where a classifier matches (is activated) if it completely matches to a posted message on the message list.

The '#' (match anything) in the conditional-part of classifier matching directly provides the capability for the system to generalize, and for the pressures on the system to change the levels of generalization. For example, (given input messages of size 5), a string of all '#' (#####) can match any possible string on the message list. The more information ('1's and '0's) in the condition, the more specific the matching behaviour. The ternary alphabet facilitates feature extraction, not limited to contiguous sequences, and it does not require a user parameter to define the matching threshold. The match/no-match decision process may result in input signals that are not matched, and thus to which the system does not respond. This may be a desired behaviour, although it is common in LCS implementations to provide catchall rules (like all '#') such that input signals are not missed.

B. *Negative Selection Algorithm*

The negative selection algorithm is an immune inspired novelty detection and classification algorithm that uses random rule generation and parameterised matching-based classification [44]. A repertoire of bitstring detectors is prepared using random generation and the deletion (negative selection) of those detectors that match known data. Thus, a class of data is modelled in the complement space of known examples. The traditional matching rule used is called the 'r-contiguous bits' matching rule in which a match between a detector and an input signal occurs if the number of matching bits between the two strings is equal or larger to the user parameter 'r'. In addition, as the name suggests, the $\geq$ r matching bits must be clustered in a contiguous region of the detector [36].

A simpler matching rule is the inverse-Hamming distance (number of matching bits), where the number of matching bits must exceed a user specified threshold (r). A simplified version of the r-contiguous bits matching rule was proposed by Balthrop, et al. called the r-chunks matching rule [14,24]. Detectors are length 'r' rather than the full length of the input message (L), further, detectors match to input messages at a given anchor position, called the window. Thus, a given r-contiguous matching detector may be replaced by a set of r-chunk detectors, providing finer granularity in matching at the cost of more detector-pattern comparisons.

The following table captures the general properties of the three more common matching rules.

Matching Rule	Feature Information	Summary
r-(inverse)Hamming	Match	Generalize input signals to features which may or may not be contiguous
r-Contiguous	Match, Region	Generalize input signals to r-bit contiguous features
r-Chunk	Match, Region, Position	Generalize input signals to r-bit contiguous features, anchored to specific positions in input signals

Table 3 - Summary of common negative selection algorithm matching rules

For a review and analysis of many different Hamming-based matching rules see Harmer, et al. [35]. For a review and comparison of the effects of the three popular binary matching rules when detectors are mapped to a real-valued space, see González, et al. [9].

IV. Discussion

Like a genetic algorithm, the genetic operators of variation may be employed, such as inversion and in particular mutation. A winner-take-all clonal selection algorithm (CSA) may be phrased as an $(1,\lambda)$ Evolution Strategy, and when the selection constraint is relaxed, it may be phrased as a (μ,λ) Evolution Strategy. The 1/L mutation heuristic may be employed in clonal selection algorithms, particularly given the optimization task is likely to be the development of feature detectors driven by Hamming distance (linear optimization function). The primary difference between a CSA and a genetic algorithm and evolution strategy is that the repertoire may be required to maintain solutions to multiple different objective functions. Thus, the CSA is more like a feature detection algorithm, a classification algorithm, or a multiple-objective optimization algorithm.

The genetic algorithm has traditionally simplified the genetics of its metaphorical inspiration, in particular the transcription of genes information to proteins. Similar simplifications are traditionally employed in artificial immune systems such that genetic operators are applied on the same representation as the functional operators like selection (matching). The ternary approach to matching provides a variable (self-tuning/pressure-based tuning) amount of generalization (and specialisation), although with the added requirement of catchall rules. The other approaches require parameterisation, and thus some idea of the size of the features to detect and in which to respond. The ternary alphabet is the most general, and may be specialised with match size, contiguous regions, and position information as required by the problem.

Acknowledgments

Tim Hendtlass for his patience and for providing useful feedback on drafts of this paper

References

[1] A. E. Eiben and C. A. Schippers, On evolutionary exploration and exploitation *Fundamenta Informaticae*, vol. 35, pp. 35-50, Aug, 1998.

[2] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In: *Genetic Algorithms and Simulated Annealing (Research Notes in Artificial Intelligence)*, Anonymous USA: Morgan Kaufmann, 1987.

[3] David E. Goldberg, Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis *Complex Systems*, vol. 3, pp. 153-171, 1989.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, USA, Canada: Addison Wesley Publishing Company, Inc., 1989.

[5] David E. Goldberg, Bradley Korb, and Kalyanmoy Deb, Messy genetic algorithms motivation, analysis, and first results *Complex Systems*, vol. 3, pp. 493-530, 1989.

[6] David E. Goldberg, K. Deb, and J. Horn, "Massive Multimodality, Deception, and Genetic Algorithms," *Proceedings Parallel Problem Solving from Nature 2, PPSN-II,* Brussels, Belgium, pp. 37-48, 1992.

[7] David Edward Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, USA: Kluwer Academic Publishers, 2002.

[8] David H. Ackley. *A connectionist machine for genetic hillclimbing*, Norwell, MA, USA: Kluwer Academic Publishers, 1987.

[9] Fabio González, Dipankar Dagupta, and J. Gómez, "The effect of binary matching rules in negative selection," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO),* pp. 198-209, 2003.

[10] G. E. Liepins and M. D. Vose, Representational issues in genetic optimization *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 2, pp. 101-115, 1990.

[11] G. Rudolph, Convergence Properties of Evolutionary Algorithms 1997.

[12] Günter Rudolph, How Mutation and Selection Solve Long Path Problems in Polynomial Expected Time *Evolutionary Computation*, vol. 4, pp. 194-205, Summer, 1996.

[13] Heinz Muhlenbein, "How Genetic Algorithms Really Work: I. Mutation and Hillclimbing," *Parallel Problem Solving from Nature 2, PPSN-II,* Brussels, Belgium, pp. 15-26, 1992.

[14] J. Balthrop, S. Forrest, and M. R. Glickman, "Revisiting LISYS: parameters and normal behavior," *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02) ,* Honolulu, HI, USA, pp. 1045-1050, 2002.

[15] J. C. Culberson, Mutation-Crossover Isomorphisms and the Construction of Discriminating Functions *Evolutionary Computation*, vol. 2, pp. 279-311, Fall, 1994.

[16] J. David Schaffer and Larry J. Eshelman, "On Crossover as an Evolutionarily Viable Strategy," *Proceedings of the 4th International Conference on Genetic Algorithms,* San Diego, CA, USA, pp. 61-68, 1991.

[17] J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," *Proceedings of the third international conference on Genetic algorithms,* George Mason University, United States, pp. 51-60, 1989.

[18] J. H. Holland. Adaptation. In: *Progress in Theoretical Biology IV*, eds. R. Rosen and F. Snell. Academic Press, 1976.pp. 263-293.

[19] J. H. Holland, Adaptive algorithms for discovering and using general patterns in growing knowledge-bases *International Journal of Policy Analysis and Information Systems*, vol. 4, pp. 217-240, 1980.

[20] Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb, "Long Path Problems," *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature,* pp. 149-158, 1994.

[21] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, USA: MIT Press, 1992.

[22] Joseph Culberson, "Genetic Invariance: A New Paradigm for Genetic Algorithm Design," University of Alberta, USA, Technical Report TR92-02, Jun 1992.

[23] Jürgen Hesser and Reinhard Männer, "Towards an Optimal Mutation Probability for Genetic Algorithms," *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature,* pp. 23-32, 1991.

[24] Justin Balthrop, Fernando Esponda, Stephanie Forrest, and Matthew Glickman, "Coverage and Generalization in an Artificial Immune System," *Proceedings of the Genetic and Evolutionary Computation Conference (CEC02),* pp. 3-10 , 2002.

[25] Kalyanmoy Deb and David E. Goldberg, "Analyzing Deception in Trap Functions," *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA),* Vail, Colorado, USA, pp. 93-108, 1993.

[26] L. Darrell Whitley, "Fundamental Principles of Deception in Genetic Search," *Foundations of genetic algorithms,* pp. 221-241, 1991.

[27] L. Davis, "Bit-climbing, representational bias, and test suite design," *Proceedings of the fourth international conference on genetic algorithms,* pp. 18-23.

[28] L. J. Eshelman, "The CHC adaptive search algorithm How to safe search when engaging in nontraditional genetic recombination," *Foundations of genetic algorithms,* pp. 265-283, 1991.

[29] Lee Altenberg, "Fitness Distance Correlation Analysis: An Instructive Counterexample," *Proceedings of the 7th International Conference on Genetic Algorithms,* East Lansing, MI, USA, pp. 57-64, 1997.

[30] M. Clergue and P. Collard, "GA-hard functions built by combination of Trap functions," *Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02,* Honolulu, HI, USA, pp. 249-254, 2002.

[31] Melanie Mitchell and John H. Holland, "When Will a Genetic Algorithm Outperform Hill Climbing?," *Proceedings of the 5th International Conference on Genetic Algorithms,* Urbana-Champaign, IL, USA, pp. 647 , 1993.

[32] Melanie Mitchell, John H. Holland, and Stephanie Forrest, "When will a Genetic Algorithm Outperform Hill Climbing," *Advances in Neural Information Processing Systems 6 (7th NIPS Conference),* Denver, Colorado, USA, pp. 51-58, 1994.

[33] Melanie Mitchell, Stephanie Forrest, and John H. Holland, "The royal road for genetic algorithms: Fitness landscapes and GA performance," *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life,* Paris, pp. 245-254, 1992.

[34] P. Collard, M. Clergue, and F. Bonnin, "Misleading functions designed from alternation," *Proceedings of the 2000 Congress on Evolutionary Computation,* La Jolla, CA, USA, pp. 1056-1063, 2000.

[35] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont, An artificial immune system architecture for computer security applications *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 252-280, Jun, 2002.

[36] Patrik D'haeseleer, S. Forrest, and P. Helman, "An immunological approach to change detection: algorithms, analysis and implications," *IEEE Symposium on Security and Privacy,* Oakland, CA, USA, pp. 110-119, 1996.

[37] Philippe Collard, Alessio Gaspar, Manuel Clergue, and Cathy Escazut, "Fitness Distance Correlation, as statistical measure of Genetic Algorithm difficulty, revisited," *European Conference on Artificial Intelligence,* pp. 650-654, 1998.

[38] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. *Learning Classifier Systems: From Foundations to Applications*, Berlin / Heidelberg: Springer, 2000.

[39] R. J. Quick , Victor J. Rayward-Smith, and G. D. Smith, "Fitness Distance Correlation and Ridge Functions ," *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature,* pp. 77-86, 1998.

[40] S. Nijssen and T. Back, An analysis of the behavior of simplified evolutionary algorithms on trap functions *IEEE Transactions on Evolutionary Computation*, vol. 7, pp. 11-22, Feb, 2003.

[41] S. W. Wilson, "GA-Easy Doe Not Imply Steepest-Ascent Optimizable," *Proceedings of the Fourth International Conference on Genetic Algorithms,* pp. 85-89, 1991.

[42] Stefan Droste, Thomas Jansen, and Ingo Wegener, "On the Optimization of Unimodal Functions with the (1 + 1) Evolutionary Algorithm," *Parallel Problem Solving from Nature — PPSN V,* pp. 13, 1998.

[43] Stefan Droste, Thomas Jansen, and Ingo Wegener, On the analysis of the (1+1) evolutionary algorithm *Theoretical Computer Science*, vol. 276 , pp. 51-81, Apr, 2002.

[44] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri, "Self-Nonself Discrimination in a Computer," *Proceedings of the 1992 IEEE Symposium on Security and Privacy,* pp. 202-212, 1994.

[45] Stephanie Forrest and Melanie Mitchell, "Relative Building-Block Fitness and the Building-Block Hypothesis," *Proceedings of the Second Workshop on Foundations of Genetic Algorithms,* Vail, Colorado, USA, pp. 109-126, 1993.

[46] T. Jones, Evolutionary Algorithms, Fitness Landscapes and Search 1995. University of New Mexico.

[47] Terence C. Fogarty, "Varying the probability of mutation in the genetic algorithm," *Proceedings of the third international conference on Genetic algorithms,* George Mason University, United States, pp. 104-109, 1989.

[48] Terry Jones and Stephanie Forrest, "Fitness Distance Correlation

as a Measure of Problem Difficulty for Genetic Algorithms," *Proceedings of the 6th International Conference on Genetic Algorithms,* Pittsburgh, PA, USA, pp. 184-192, 1995.

[49] Thomas Back, "Optimal Mutation Rates in Genetic Search," *Proceedings of the Fifth International Conference on Genetic Algorithms,* pp. 2-9, 1993.

[50] Thomas Back , David B Fogel, and Zbigniew Michalwicz. *Evolutionary Computation 1 - Basic Algorithms and Operators*, Bristol, UK: Institute of Physics (IoP) Publishing, 2000.

[51] William M. Spears, "Crossover or Mutation?," *Proceedings of Foundations of Genetic Algorithms Workshop,* pp. 221-237, 1992.

[52] William M. Spears, The Role of Mutation and Recombination in Evolutionary Algorithms 1998. George Mason University.