Journal of Al-Nahrain University Vol.14 (2), June, 2011, pp.199-206 Science

Breaking Knapsack Cipher Using Population Based Incremental Learning

Nasreen J. Kadhim
Computer Science Department, College of Science, Baghdad University.
E-mail: Nasreen_jawad@yahoo.com.

Abstract

The demand for effective internet security is increasing exponentially day by day. Businesses have an obligation to protect sensitive data from loss or theft. Such sensitive data can be potentially damaging if it is altered, destroyed or if it falls into the wrong hands. So they need to develop a scheme that guarantees to protect the information from attacker. Cryptology which is the science and study of systems for secret communication is at the heart of providing such guarantee. Breaking knapsack cipher using Population Based Incremental Learning (PBIL) is suggested in this paper. Results of this implementation are compared with results of Poonam Grag, Aditya Shastri, and D.C. Agarwal (2006) which had made an enhancement on the efficiency of genetic algorithm attack on knapsack cipher suggested by Spillman in 1993. The result of comparison shows that Population Based Incremental Learning has less complexity and enjoys high speed (i.e. correct results will be generated with less number of generation compared to that generated using genetic algorithm) also the overhead for genetic algorithm operations is significantly higher than for Population Based Incremental Learning.

1. Introduction

Cryptology is the science of building and analyzing different encryption and decryption methods. Cryptology consists of two subfields; Cryptography and Cryptanalysis. The basic aim of Cryptography is to allow the intended recipients of a message to receive the message properly while preventing eavesdroppers from understanding the message. Cryptanalysis is the science and study of method of breaking cryptographic techniques i.e. ciphers. In other words it can be described as the process of searching for flaws or oversights in the design of ciphers [1],[2].

The application of genetic algorithms in the cryptanalysis of knapsack ciphers is suggested by Spillman [3] in 1993. The efficiency of genetic algorithm attack on knapsack cipher is then enhanced and re-implemented with variation of initial assumption by Poonam Grag, Aditya Shastri, and D.C. Agarwal [2] in 2006.

One of first knapsack cipher was proposed by Markle and Hellman in 1975 which utilized a NP_Complete problem for its security. The Markle-Hellman knapsack cipher encrypts a message as a knapsack problem. The plaintext block transforms into binary string (the length of block is equal to the number of elements in knapsack sequence). One value determines that an element will be in the target sum. This sum is a ciphered message [2]. Table 1 shows an example of solving knapsack problem for the entry numbers sequence: 1 3 6 13 27 and 52.

Table (1)
Example of Knapsack Encryption.

Plaintext	*Knapsack sequence*	*Ciphertext*
111001	1 3 6 13 27 52	1+3+6+52= 62
010110	1 3 6 13 27 52	3+13+27 = 43
000001	1 3 6 13 27 52	52 = 52

The public/private key aspect of this approach lies in the fact that there are actually two different knapsack problems referred to as the easy knapsack and hard knapsack. The Markle-Hellman algorithm is based on this property. The private key is a sequence of numbers for a superincreasing knapsack problem. The public key is a sequence of numbers for a normal knapsack problem with the same solution [1], [2].

Easy knapsacks have a sequence of numbers that are superincreasing- that is, each number is greater than the sum of previous numbers in the sequence. The knapsack solution with the superincreasing sequence proceeds as follows: The target sum is compared with a greatest number in the sequence. If the target sum is smaller than this number, the knapsack will not fill, otherwise it will. Then the smaller element is subtracted from the target sum, and the result of subtraction is compared with next element. Such operation is done until the smallest element of sequence is reached. If the target sum is reduced to 0 value, then solution exists. In the other case solution does not exists. The sequence of 0 and 1 resulting from this operation gives the plaintext. The superincreasing knapsack is easy to decode, which means that it does not protect the data. Anyone can recover the bit pattern from the target sum for a superincreasing knapsack if the elements of the superincreasing knapsack are known.

Markle and Hellman suggested that such a simple knapsack be converted into a trapdoor knapsack which is difficult to break. The algorithm works as follows:

1. Select a simple knapsack sequence. Elements make a superincreasing $A^{'} = (a_1^{'}, a_2^{'}, ..., a_n^{'})$.
2. Select an integer value *u* greater than sum of all elements of superincreasing sequence.
3. Select another integer *w* that the gcd(*u,w)=1*, that is number *u* and *w* are reciprocally prime.
4. Find the inverse of $w \bmod u - w^{-1}$.
5. Construct the hard knapsack sequence $A = wA' \bmod u$ i.e. $a_i = wa_i^{'} \bmod u$.

The trapdoor sequence *A* could be published as a public key (encryption key). The private (secret) key for this cipher consists of a simple knapsack sequence *A'*, so-called trapdoor, values u, w, w^{-1}.

The encoding is done as follows: The message divides into *n* bits blocks (each block contains as many elements as simple knapsack sequence). Values in the message block shows that the element will be in the target sum. The target sum of each block is ciphertext [2].

This paper suggests the application of Population Based Incremental Learning (PBIL) in the cryptanalysis of knapsack ciphers.

This section: introduced an overview of the problem, the field of science that we will work on it, aims and objectives to be achieved.

Section 2: introduces an overview of PBIL together with the process of updating the probability vector and its control parameters.
Section 3: details PBIL algorithm used for the cryptanalysis process.
Section 4: presents a comparison between experimental results obtained using PBIL and results obtained from [2] together with a discussion. Also examines the effects of changing control parameters of the PBIL algorithm.
Section 5: discusses the applicability of the results reported in section 4 and concludes with a discussion of the benefits of using PBIL in comparison to standard genetic algorithms for breaking knapsack cipher.

2.Introduction to Population Based Incremental Learning

Population Based Incremental Learning (PBIL) is a simple version of the estimation of distribution algorithm (EDA), the method was first proposed by by Baluja [4] around 1994 for single objective optimization [5]. It is a technique derived by combining aspects of genetic algorithms and competitive learning. This algorithm and its variants have been shown by Baluja to significantly outperform standard GA approaches on a variety of stationary optimization problems ranging from toy problems designed specifically for GA's to NP-complete problems. It has also been applied to various real-world applications including autonomous highway vehicle navigation. PBIL's most significant difference from standard genetic algorithms is the removal of the *population* found in GA's [6]. [7] PBIL is a statistical approach to evolutionary computation in which a solution is represented by a fixed length binary vector and a probability vector is maintained to sample the new solutions. PBIL is a genetic algorithm that can be used to perform a guided random search. The guided random/stochastic search method is a randomized search in which attention is adaptively increased to focus on the band combinations that return promising results. PBIL is a combination of iterary and evolutionary optimization methods. This algorithm is based on Genetic Algorithm (GA) mechanisms along with weight updating rules in supervised competitive learning.

Generally speaking, PBIL combines the elements of Genetic Algorithms and Reinforcement Learning [8]. Unlike, GAs where operations are applied on the population itself, in PBIL the operations are applied on the probability vector. The basic representation of a solution can be the same as in a GA but instead of storing each possibility explicitly the population is replaced by a probability distribution. The essence of PBIL is to represent different solutions with a single probability vector and hence maintain the diversity in the population to some extent. It is evident that after successive generations the probability vector either converges to 0 or 1 and hence algorithm may trap in local optima as the similarity in the vectors generated increases- a similar dilemma faced by the simple evolutionary algorithms. But the good feature of the PBIL is that it can be controlled explicitly by a learning rate which can greatly affect the speed of convergence [9]. During each generation, sampling the vector generates a number of potential solutions. Every one of these solutions is then evaluated according to the fitness function. The probability vector is pushed towards the solution vector with highest fitness; it is also pushed towards the complement of the solution with lowest (worst) fitness. After updating the probability vector, a new set of solution vectors is produced, and the cycle continues [4]. Furthermore, a special mutation operation is used when updating the vector: each position in the probability vector is shifted to a random direction with a small probability. As in simple evolutionary algorithms mutation plays an important role in the later stages to maintain the diversity in the population [9]. For PBIL, there are two ways of defining a mutation operator. The first is to perform the mutation directly on the vectors generated. The second method is to perform a mutation on the probability vector; this mutation can be defined as a small probability of perturbation on each of the positions in the probability vector. Both of these forms of mutation have the same effect as mutation in standard genetic algorithms: to help preserve diversity. For the experiments conducted in this paper, the second form of mutation is implemented [4]. Features of PBIL can be summarized as follows [10]:

- It has no crossover and fitness proportional operators.
- It works with probability vector (number in range 0-1). This probability controls the random bitstrings generated by PBIL.
- The probability vector is used to create other individuals through learning.
- There is no need to store all solutions in the population. Only two solutions, the current best solution and the solution being evaluated, and the probability vector are stored.

2.1. Probability Vector Updating

The probability vector used in PBIL can be assumed as a template for generating solution vectors. Initially every element of this probability vector is set to 0.5 and then on the basis of selected promising solutions, the probability vector is modified according to the Hebbian rule by using the fittest population member. Since, PBIL uses the fittest individual of a population to update the probability vector therefore, it is expected that after a number of generations the probability vector will be shifted towards good solutions [9]. [4] The process of updating the probability vector in PBIL algorithm is summarized in the following equation [11]:

$$P^{(t+1)} = P^{t} \times (1.0 - lr) + SV^{t} \times lr \quad ....(1)$$

where lr, the learning rate, is $0 < lr \leq 1$; SV^{t} is selected individual.

It should be noted that the probability vector not only specifies the prototype based upon the high evaluations of the sample solutions, but also guides the search, which produces the next sample point from which to “learn”.

The classes of problems to be addressed by PBIL are unlike many other problems for which competitive learning (CL) is often employed. In many domains to which CL is applied, one of the largest difficulties in training the CL-network is the lack of available training data. However, there is an abundance of training data in the class of problems attempted here. Training data is available through the evaluation of potential solution vectors. Nonetheless, to be efficient, the algorithm must minimize the number of function evaluations performed. Therefore, as information regarding the characterization of high evaluation vectors becomes available, it is incorporated into the search; the updated probability vector is used to generate the next population of sample points [4].

2.2. Control Parameters

There are four parameters which can be adjusted in a PBIL algorithm. These include population size, learning rate, mutation rate and mutation shift. There are some variants of PBIL which also use learning from negative samples and hence negative learning rate also becomes an important parameter. Settings of these parameters have a significant impact on the performance of a PBIL algorithm and normally these parameters are set empirically. In PBIL, “the learning rate has a direct impact on the trade-off between exploration of the function space and exploitation of the exploration already conducted”. Population size is a critical control parameter in any evolutionary algorithm. A large population size implies a slow convergence and vice-versa. In most evolutionary algorithms population size is determined heuristically and is kept constant throughout the evolution process. Mutation operator helps in maintaining the diversity in the population at later stages of an Evolutionary Computation algorithm and guarantees (at least theoretically) to explore the whole population in order to discover the global optimum. The mutation shift is the magnitude of the effect of mutation on the probability vector and its value is normally kept small enough to provide small perturbation.

Mutation of the probability vector will be according to the following equation [9],[11]:
If random (0,1] < β

$$P^{(t+1)} = P^{t} \times (1 - m) + random(0\,|\,1) \times m \quad \text{.......}(2)$$

where, β is probability of mutation occurring in each position and m is the mutation rate.

3. Cryptanalysis of Knapsack Cipher

The cryptanalysis starts from cipher text, which has an integer form. Each number represents a target sum of hard knapsack problem. The goal of the PBIL algorithm is to translate each number into the correct knapsack, which represents the ASCII code for the plaintext characters.

Encoding

There are certain restrictions on the encoding algorithm:

- Only the ASCII code will be encrypted.
- The superincreasing sequence will have 8 elements, these number of elements guarantee that each character has a unique encoding (there are 256 ASCII codes and 8 elements length will allow to encrypt 2^8 characters).

Initialization

Initially every element of probability vector is set to 0.5 to ensure uniformly-random bitstrings. The number of bits in the probability vector is equal to the number of elements key (i.e., 8).

Generate Samples

Generate a population of uniformly-random bitstrings then generate solution vectors according to probabilities in the probability vector.

Evaluation

Spillman proposed the fitness measure given in equation (3) below [3].

$$Fitness = \left.\begin{matrix} 1-\left(\dfrac{\text{target} - \text{sum}}{\text{target}}\right)^{1/2} & \text{if sum} <= \text{target} \\ 1-\left(\dfrac{\text{target} - \text{sum}}{\text{maxdiff}}\right)^{1/6} & \text{if sum} > \text{target} \end{matrix}\right\} \quad \text{..................................}(3)$$

Let $M = (m_1, m_2, \ldots\ldots, m_n), m_i \in \{0,1\}$ be an arbitrary solution and the public key $A = (a_1, a_2, \ldots.., a_n)$

Sum= $\sum_{i=1}^{n} a_i m_i$

FullSum= $\sum_{i=1}^{n} a_i$

MaxDiff= max(Target, FullSum-Target).

The fitness of the solution vectors will be evaluated according to the above objective function. The fitness value evaluates how the given sum is close to the target value for the knapsack. If the value of sum is greater than targets then it will produce the infeasible solutions else it will produce a high fitness value and produce feasible solutions. Feasible solutions have a greater chance of being followed by the algorithm.

Selection

Find the fittest solution. The "best" individual is used to update the probability vector so that the probability of producing solutions similar to the current best individuals is increased. Optimal solutions resulted when the number of best individuals used to update the probability vector decreases.

Update probability vector

Update the probability vector by using the fittest solution according to equation (1).

Mutation

Mutate the probability vector according to equation (2). The mutation process moves between two random points. It helps to prevent the algorithm from being stuck in a local optimal point.

Stop condition

A new generation is created and the steps are repeated until a satisfactory solution is found or 200 generation is performed.

<u>Example:</u>

This example shows results obtained from the execution of part of cryptanalysis knapsack cipher using PBIL. Starting from ciphertext, **Ciphertext:** 128821**, Character:** 'O', **Generation Size** (number of samples): 4

Initialize Probability Vector *P*:

$$[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$$

Generate Random Samples :

$$\begin{matrix} 0.8147 & 0.9580 & 0.1270 & 0.9134 & 0.6324 & 0.0975 & 0.2785 & 0.5469 \\ 0.9575 & 0.9640 & 0.1576 & 0.9706 & 0.9572 & 0.4854 & 0.8003 & 0.1419 \\ 0.4218 & 0.9157 & 0.7922 & 0.9595 & 0.6557 & 0.0357 & 0.8491 & 0.9340 \\ 0.6787 & 0.7577 & 0.7431 & 0.3922 & 0.6555 & 0.1712 & 0.7060 & 0.0318 \end{matrix}$$

Solution Vectors Generated According to Probabilities in *P*:

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{matrix}$$

Evaluate Fitness for Solution Vectors:

$$[0.4620 \quad 0.6688 -0.19131 \quad 0.3818 \quad 0.8187]$$

Then vector with the fittest solution which is the fourth solution vector in our example with fitness (0.8187) will be chosen to update *P* according to equation (1), then mutation will be done on *P* according to equation (2), and the steps are repeated until the correct knapsack, which represents the ASCII code for character 'O' is obtained or 200 generation is performed..

4. Results and Discussion

An experimental result for PBIL was generated with 5 runs per data point using MATLAB. PBIL is run with the mentioned algorithm for each target sum mentioned in Table (2). Each attack is run 5 times with constant entry parameters as shown in Table (2) and then results are averaged.

Table (2)

Samples in each generation	75
Learning Rate	0.1
Mutation Probability	0.11
Mutation Shift (Mutation Rate)	0.02

The 8-element sequence of hard knapsack problem (21031 63093 15371 11711 23422 58555 16615 54322) is used to encode 8 bit ASCII code. This sequence has been created from superincreasing sequence (1 3 7 13 26 65 119 267), u is equal to 65423 and w integer equal 21031 ($w^{-1} = 5363$). The MACRO word has been encrypted. Table 3 shows the result of encoding.

Table(3)
Encryption By Knapsack.

Character	*ASCII Code*	*Target Sum (Ciphertext)*
M	10110010	65728
A	10000010	37646
C	11000010	100739
R	01001010	103130
O	11110010	128821

By using the same parameters as in [2], Table (4) shows the experimental results generated by PBIL; AvgGen is the average of number of generation needed after 5 runs.

Table (4)
Experimental results obtained using PBIL.

Character	*Exp 1*	*Exp 2*	*Exp 3*	*Exp 4*	*Exp 5*	*AvgGen*
M	15	16	10	15	10	13.2
A	11	10	18	12	12	12.6
C	11	9	11	11	12	10.8
R	17	12	11	12	11	12.6
O	14	11	9	13	10	11.4
Sum						12.1

Table (5)
Experimental results obtained from [2].

Character	*Exp 1*	*Exp 2*	*Exp 3*	*Exp 4*	*Exp 5*	*AvgGen*
M	13	2	68	1	1	17
A	226	67	1	1	265	112
C	173	48	1	279	853	271
R	290	4	108	44	1	89.4
O	2	1	210	1	222	87.2
Sum						115

Table (4) shows that on an average of 12.1 generations is enough for reaching to the correct results. These results are analyzed and compared with results in [2]. [2] Gives always correct results similar to results as obtained by us on an average of 115 populations (i.e. 115 generation) as illustrated in table 4 above. The area of possible results in [2] and in our work is 2^8 .

Now a discussion to the effect of control parameters on producing correct results will be presented here by changing one of the parameters of Table (2) and leave the others constant:

Changing the learning rate have been shown in tables 6 and 7:

Table (6)
Experimental results obtained at lr = 0.3.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	2	2	4	2	2
A	2	6	4	5	2
C	4	2	4	2	4
R	5	2	6	5	2
O	5	2	2	2	4

Table (7)
Experimental results obtained at lr = 0.03.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	49	54	48	57	46
A	52	50	60	43	46
C	42	60	63	48	52
R	45	70	55	55	53
O	74	65	58	61	59

It is obvious that increasing the learning will lead to correct results in less number of generations than results generated in Table (4), while more number of generations than in Table (4) will be required to produce the same results when learning rate decreases.

Table (8) will present the effect of decreasing generation size:

Table (8)
Experimental results obtained when generation size (i.e. number of samples) = 25.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	22	42	23	22	26
A	29	30	32	24	20
C	20	27	27	20	42
R	20	18	27	32	15
O	29	27	32	32	23

Results in Table (8) above show that results became worse when generation size decreases.

Tables (9) and (10) present the effect of changing the mutation shift control parameter:

Table (9)
Experimental results obtained at mutation shift = 0.3.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	69	186	91	184	62
A	41	29	40	55	56
C	50	86	28	107	74
R	31	169	40	53	40
O	49	64	110	57	77

Table (10)
Experimental results obtained at mutation shift = 0.03.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	12	10	13	11	8
A	12	11	12	13	11
C	12	8	12	10	10
R	10	12	12	10	13
O	10	10	12	13	8

Results in Tables (9) and (10) show that better results will be obtained by reducing mutation shift control parameter.

The effect of changing the mutation probability control parameter is shown in Tables (11) and (12):

Table (11)
Experimental results obtained at mutation probability = 0.8.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	21	16	21	25	23
A	21	19	23	22	25
C	24	22	21	25	35
R	41	24	19	23	24
O	23	26	20	29	47

Table (12)
Experimental results obtained at mutation probability = 0.01.

Character	*Exp1*	*Exp2*	*Exp3*	*Exp4*	*Exp5*
M	10	10	12	10	12
A	10	13	12	10	10
C	10	10	13	10	12
R	10	8	10	10	10
O	8	12	10	13	8

Tables (11) and (12) show that decreasing Mutation probability will produce correct results in less number of generations than results generated in Table (4).

The above results indicate that compared to GA, PBIL has less complexity and enjoys high speed (i.e. less number of generations required to produce correct results) also the overhead for genetic algorithm operations is significantly higher than for PBIL.

5. Conclusion

Population Based Incremental Learning is presented in this paper for breaking knapsack cipher. This paper indicates that the use of PBIL instead of genetic algorithm gave us an efficient results (less number of generations required to produce correct results). Large generation number, large generation size, decreasing the number of selected individuals used for updating the probability vector, high learning rate, small mutation probability and small mutation shift are the factors for producing an efficient solutions. The results became worse when learning rate, number and size of generation decrease and when the coefficient mutation and the number of selected individuals increase. In our experiments average of 12.1 generations gives the correct result. In [2,,8] an average of 115 generation gives the correct result. PBIL accelerates the speed of the algorithm forward the correct solution and are attained faster than genetic algorithm, both in terms of the number of evaluations performed and the clock speed.

References

[1] Schneier B., "Applied Cryptography", 2nd edition, John Willy &Sons, New York 1996.

[2] Poonam Grag, Aditya Shastri, and D.C. Agarwal, "An Enhanced Cryptanalytic Attack on Knapsack Cipher Using Genetic Algorithm". Transactions on Engineering , Computing and Technology V12 March 2006 ISSN 1305-5313.

[3] Spillman R. "Cryptanalysis of Knapsack Ciphers Using Genetic Algorithms" Cryptologia, 17(4): 367-377, October 1993.

[4] Baluja, S. (1994). "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning". Technical Report CS-94-163, Carnegie Mellon University, School of Computer Science.

[5] T. Kunakote and S. Bureerat**.** "Topology Optimization of a Plate Structure Using Multiobjective Population - Based Incremental Learning". Technology and Innovation for Sustainable Development Conference (TISD2008) Faculty of Engineering, Khon Kaen University, Thailand 28-29 January 2008.

[6] F. Southey and F. Karray, "Approaching Evolutionary Robotics Through Population-Based Incremental Learning". University of Waterloo, http://backtrack.uwaterloo.ca.

[7] Marten Pelikan, David E. Goldberg, Fernando G. Lobo. "A Survey of Optimization by Building and Using Probabilistic Models". Computational Optimization and Applications, 21, 5–20, 2002.
[8] Timothy Gosling, Nanlin Jin and Prof. Edward Tsang (2004). "Population Based Incremental Learning versus Genetic Algorithms: Iterated Prisoners Dilemma". Technical Report CSM-401 (March 2004).
[9] Imtiaz H. Khan. "A Comparison of ‘Evolutionary Algorithm with Guided Mutation’ With ‘Population Based Incremental Learning’ And ‘Compact Genetic Algorithm’. M.Sc. Thesis, Department of Computer Science, University of Essex (September 2005).
[10] KA. Folly. "Application of Evolution Algorithm in Power System Control Design". IEEE PES PowerAfrica 2007 Conference and Exposition, Johannesburg, South Africa, 16-20 July 2007.
[11] S. Bureerat and K. Sriworamas. 2007. "Population-Based Incremental Learning for Multiobjective Optimization. Advances in Soft Computing", 39:223-232.

الخلاصة

يزداد الطلب يوما بعد يوم على شبكة معلومات عالية الأمنية كما أن كافة الأعمال عليها التزام لحماية البيانات الحساسة من السرقة أو الفقدان. مثل هذه البيانات الحساسة ممكن أن تدمر اذا تم تغييرها أو أن تقع في الأيدي الخاطئة ولذلك هم بحاجة لتطوير اسلوب يضمن حماية هذه المعلومات من المقتحمين. مثل هذا الضمان ممكن أن يتحقق عن طريق علم التشفير. حيث أن علم التشفير هو العلم الذي يعمل على دراسة الأنظمة الخاصة بامنية الأتصالات. في بحثنا هذا تم اقتراح التعلم المتنامي المستند على عدد السكان لكسر شفرة نابساك ثم تمت مقارنة نتائج هذا البحث مع نتائج بحث Poonam Grag, Aditya Shastri, D.C. Agarwal (2006) والذي تم فيه تحسين كفاءة الخوارزميات الجينية في عملية تحليل الشفرة لشفرة نابساك والمقترح من قبل Spillman في (1993). نتائج التجارب التي تمت تأديتها في هذا البحث سوف تظهر أن التعلم المتنامي المستند على عدد السكان مقارنة بالخوارزميات الجينية فيه تعقيد أقل ويتمتع بسرعة عالية (أي من الممكن الحصول على نتائج صحيحة بعدد أجيال أقل مما في الخوارزميات الجينية) وكذلك فأن السقف لعمليات الخوارزميات الجينية هو أعلى بشكل مؤثر مما في التعلم المتنامي المستند على عدد السكان.