A Sensitivity Analysis of Evolutionary Algorithms in Generating Secure Configurations

Shuvalaxmi Dass
Computer Science Department
Texas Tech University
shuva93.dass@ttu.edu

Akbar Siami Namin
Computer Science Department
Texas Tech University
akbar.namin@ttu.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378307

***Abstract*—The growth of Cyber-physical Systems (CPS) has been increased in recent years. This has led to the coupling of highly complex cyber-physical components. With the integration of such complex components, new security challenges have emerged. Studies involving security issues in CPS have been quite difficult to be generalized due to the presence of heterogeneity and the diversity of the CPS components. These systems are subject to various vulnerabilities, threats and attacks, as a consequence of complex versions of CPS being introduced over time. This paper deals with vulnerabilities caused due to improper configurations in the software component of cyber-physical systems. Evolutionary algorithms such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) can be employed to adequately test the underlying software for certain categories of vulnerabilities. This paper provides a detailed sensitivity analysis of these evolutionary algorithms in order to find out whether changing parameters involved in tuning these algorithms affect the overall performance. This analysis is based on the estimate of the number of generation of secure vulnerability pattern vectors under the variation of different parameters. The results indicate that while there is no evidence of influential parameters in Genetic Algorithms (i.e., mutation rate and population size), changes in the parameters involved in Particle Swarm Optimization algorithms (i.e., velocity rate and fitness range) have some positive impacts on the number of secure configurations generated.**

***Index Terms*—Cyber-Physical Systems, security, Genetic Algorithm, Particle Swarm Optimization, Sensitivity Analysis.**

I. Introduction

Cyber-Physical Systems (CPS) are an integration of computation, networking, and physical devices in a closed form [11]. It incorporates various interconnected system that communicate with each other to monitor and control real IoT-related physical objects and processes. These systems are evolving continuously to cater to our daily needs in life. As some useful applications where CPSs are deployed successfully are electrical power grids, oil and natural gas distribution, transportation systems, health-care devices, household appliances, and many more. These systems also find their applications in the infrastructures of critical nature such as life support devices. On the other hand, these CPS-based platforms are becoming the target of adversarial attacks. There are constant threats to security of such systems given what is at stake. Unfortunately, it is practically impossible for such real-world systems to be free of vulnerabilities and resistant to all types of attacks.

CPS is composed of mixture of different hardware and software components. Hardware components include sensors, actuators, and embedded systems whereas software components include varied collections of proprietary and commercial third-party software products which are in charge of control and monitoring. The presence of such heterogeneous components which involves interaction between complex characteristics often put the privacy and security of the system at risk, thereby exposing vulnerabilities to exploitation. These vulnerabilities becomes difficult to assess, as new security issues arise or additional software/hardware components are integrated with the system.

Moving Target Defense (MTD) [19], [20] is an evolving defense technique through which the attack surface of the underlying Cyber Physical System is continuously changing. The deriving idea is to limit the time allotted and thus increase the cost for attackers to collect data about the target system during the reconnaissance stage of attack. Once the attack surface is changed, it becomes very expensive to attackers to redo the reconnaissance stage of attacks. However, the major research question is how the attack surface should be changed automatically with minimum cost while preserving the security of the systems. A feasible and economical solution is to change the configuration of underlying CPS with the objective of tighten the security of the system in a newly suggested configuration.

This paper focuses on generating secure configuration for the aforementioned problem. Here, we extend the initial idea of our paper [6], in adapting evolutionary algorithms on *vulnerability coverage*, to explore it further through Sensitivity Analysis (SA). In our earlier work [6], we demonstrated that how evolutionary algorithms can be utilized in generating a set of secure configurations using CVSS (Common Vulnerability Scoring System) as the fitness metric.

In this paper, our objective is to study whether certain parameters in these evolutionary algorithms are playing more critical roles in generating secure configurations. We conduct sensitivity analysis to assess the impact of independent variables on dependent variable under certain specific and controlled conditions. In other words, sensitivity analysis is a means to quantify the significance of underlying model's parameters on the behavior of the system [8]. This paper makes the following key contributions:

– We conduct a sensitivity analysis on parameters involved in evolutionary algorithms to study the significance of these parameters on the number of secure instances generated by these evolutionary algorithms.
– We observed that the parameters involved in Genetic Algorithms such as mutation rate and population size have little to no impact on the performance of genetic algorithms.
– Unlike GAs, we observed that the parameters involved in Particle Swarm Optimization (PSO) play an important role in the generation of number of instances of secure configurations.

The remainder of this paper is organized as follows: Section II reviews the related research work in this line of research. Section III technical background of evolutionary algorithms are presented. In Section IV, we briefly discuss our earlier work related to the problem posed and targeted in this paper. Section V presents our methodology in conducting sensitivity analysis on the evolutionary algorithms studied. Section VI presents the results of the analysis. Section VII concludes the paper and highlights the future research directions.

II. Related Work

Evolutionary algorithms have been widely used in the field of security. Crouse and Fulp [4] developed Moving Target Defense (MTD) platforms using Genetic Algorithm to ensure security in computer systems. They introduced temporal and spatial diversity in computer configuration parameters to strengthen their security. The authors carried out experiments conducting MTD using multiple computers initially configured with extremely vulnerable settings. They implemented GA with chromosome pool size, crossover and mutation rates set as 10, 0.08 and 0.02, respectively. Through this experiment, they reported that both average temporal and spatial diversity first increase and then decrease and remain constant with increase in number of iterations.

Crouse and Fulp [5] improved their MTD approach further by introducing a modified variation of GA-based MTD technique and proposed a new approach called *chromosome pool management*. This technique was introduced to address the issue of stagnancy in the pool of configurations. This stagnancy was caused due to lack of changes occurring in the set of configurations over a period of time. This approach overcomes the stagnancy challenge by reducing the fitness (security) of aging configurations by a decay value based on the time since they were last active. This approach ensures the weak configurations are eventually replaced by more secure ones as number of iterations passes.

John and Furp [9] compared and contrasted two different genetic algorithms to conduct Moving Target Defense, namely GA+PVM and GA+PDM. In GA+PVM, mutation operator mutates the parameter values based on its type (integer, option, and bit); Whereas, in GA+PDM, mutation operator changes the domain of the parameter using machine learning algorithms by removing the insecure setting from the parameter's domain.

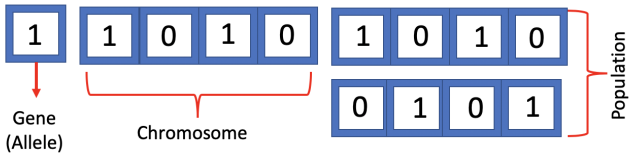


Fig. 1: The elements of a typical genetic algorithm.

Various types of sensitivity analysis is used in a wide range of fields, ranging from biology and geography to economics and engineering. It is also used in the field of security.

Syamsuddin [17] employed sensitivity analysis using several "what-if" scenarios to measure the consistency of the evaluation of their proposed model called Ternary Analytic Hierarchy Process (T-AHP) in order to help managers make strategic evaluation related to information security issues.

Do et al. [10] conducted sensitivity analysis of the finite moving averages test, which was used for detecting cyber-physical attacks on Supervisory Control And Data Acquisition (SCADA) systems. This analysis calculates the likelihood of wrong decisions with respect to variation of operational parameter. These results are then utilized to detect an attack scenario on an SCADA water network.

III. Background

This section provides background knowledge on the two different optimization algorithms: 1) Genetic and evolutionary algorithms, 2) Particle swarm optimization. We also discuss CVSS concept for the vulnerability pattern generation targeting certain level of CVSS score, as a fitness function.

A. Genetic Algorithms (GA)

Before describing the mechanism of genetic evolutionary algorithms, let us review the related terminologies in this context. As Figure 1 illustrates, a genetic algorithm involves of the following elements:

1) *Gene.* It is a single cell in a chromosome, which stores one bit of information, called an *allele*.
2) *Chromosome.* It is a collection of genes. The fitness of each chromosome is evaluated by a fitness function metric.
3) *Fitness Function.* This function is dependent on what problem domain is GA being applied on. The function gives out a fitness score to the chromosome, which shows the ability of an individual to "*compete*" in the generation.
4) *Population.* It is a collection of chromosomes. Also called *pool of configurations*. The population gives rise to one generation.

Genetic Algorithm (GA) is a search-based optimization technique that represents one branch of the field of study called evolutionary computation. The core functionality of GA is that they mimic biological process of evolution which comprises

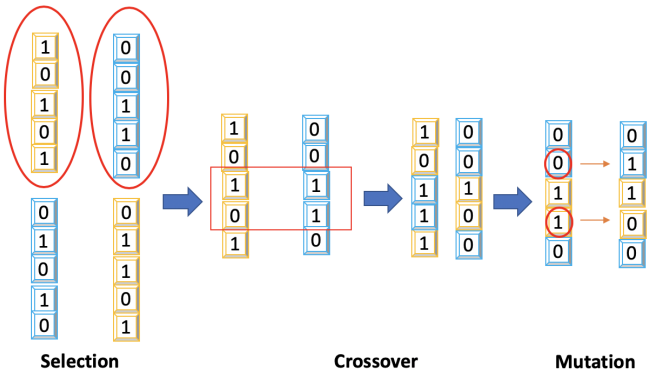


Fig. 2: Genetic algorithm process.

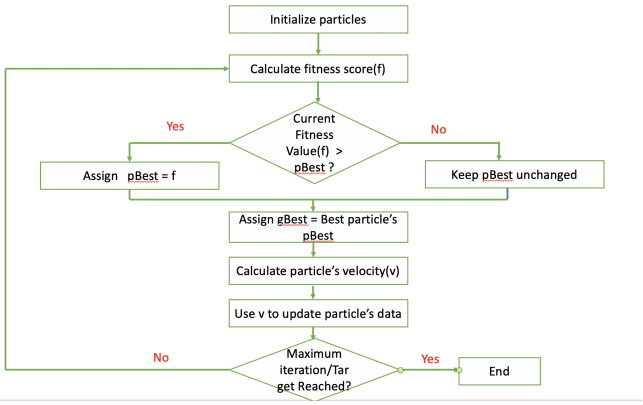


Fig. 3: The PSO's general strategy.

of reproduction and natural selection to solve for the "*fittest*" solutions [2].

The concept of GA in programming is inspired by Darwin's theory of evolution where the algorithm seeks for optimal or near-optimal solutions to the system by taking an initial population of individuals and genetically breeding them over a series of iterations [1]. This breeding process comprises of three main operators: *selection*, *crossover*, and *mutation*.

A typical GA algorithm starts with choosing a random set defined as the initial population of individuals or chromosomes, which is made up of genes. A fitness score metric to calculate the fitness of each individual is also defined [14]. During each generation, the *selection* operator chooses the fittest individuals called as parents from the current population to be part of the population formed in the next generation. The *crossover* operation then takes place between the chosen fittest parents to produce an off-spring, which will carry the best traits from both parents. Once the off-springs are born, new traits are introduced in them through *mutation*. These processes are repeated until a fixed number of generations is reached or when an acceptable fitness level has been reached for the last population [13]. Figure 2 shows the pictorial representation of the breeding process.

B. Particle Swarm Optimization (PSO)

PSO is another widely used optimization technique based on the social behaviour of bee swarms and bird flocking. In PSO, concept of swarm is analogous to that of population in GA. Similarly, a solution/individual in PSO is called a particle analogous to a chromosome in GA.

Even though PSO belongs to a different branch of evolutionary computation techniques from GA, they do share some similarities in their functionalities. For instance, both algorithms start off with a random set of initial population/solutions until a fixed number of generations is reached or when an acceptable fitness level has been reached for the last population [12]. In contrast to GA, there is no concept of crossover and mutation operations to update the particles. Modification of the particles in PSO is instead achieved by directing them towards the global optimum by their personal best position along with the swarm's best position in the search space. This makes PSO easier to implement in contrast to GA as it has comparatively fewer parameters to adjust [15].

Taking the bird flocking scenario into consideration [18], each bird in the flock represents a particle and the flock is represented by swarm. Each particle in the swarm has 1) data, which represents a possible solution, 2) velocity, which indicates how much of the data can be modified, and 3) the fitness value associated to it evaluated by the fitness function. These parameters guide the particles (i.e., birds) towards the global optimum by updating swarms. In every iteration, each particle is updated based on two best scores: 1) personal best fitness score it has achieved so far, and 2) the global best fitness score achieved by any particle in the swarm so far. Figure 3 shows the general strategy of PSO where `pBest` and `gBest` are the local and global best fits, respectively.

C. CVSS

The Common Vulnerability Scoring System (CVSS) is a free and open framework to measure the severity and the characteristics of vulnerabilities present in an affected system and calculates its severity score. It comprises of three metric groups: *Base*, *Temporal*, and *Environmental*, each representing different qualities of the vulnerability. The base group represents the intrinsic qualities whereas the temporal and environmental groups represent time-changing properties and user's environment specific properties of vulnerabilities respectively.

This paper only deals with the Base group metric. The base metric further consists of two sub-metric fields: Exploitability metrics and the Impact metrics. These sub-metrics constitute of various vector fields like AV, AC, etc., which contribute in scoring a vulnerability. The CVSS base score ranges from 0.0 to 10.0. As we go higher, the severity of vulnerability gets higher. More information on CVSS can be found at their official website[1]. Figure 4 shows an instance of CVSS vector represented by *CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H* where

[1]https://www.first.org/

Fig. 4: Example of a CVSS pattern.

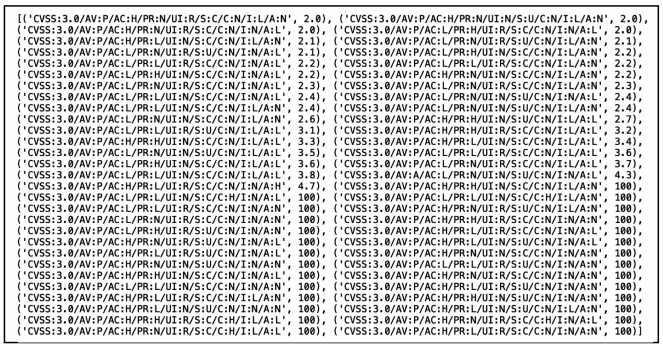


Fig. 5: A snapshot of CVSS vector patterns with their scores.

CVSS score is 7.8 denoting the severity of the vulnerability is high[2].

IV. Our Previous work

This section quickly briefs upon the methodology employed on vulnerability coverage in our previous papers [6], [7]. In our previous work, we proposed the concept of "*vulnerability coverage*" as an adequacy criterion for testing the wide-ranging vulnerabilities present in the underlying software applications. The main goal is to identify a set of vulnerability patterns that belong to a certain level of CVSS score. In order to achieve this, the key idea is to utilize Common Vulnerability Scoring System (CVSS) as a fitness metric. We implemented the two evolutionary algorithms, GA and PSO, to generate a set of vulnerability patterns for adequacy testing of underlying system. Once we obtain the pool of vulnerability patterns, we choose the representative sets of vulnerabilities with similar vulnerability vector patterns for further inspection of the system under test. Figure 5 shows a snapshot of vulnerability patterns generated by GA along with their CVSS scores. The following section extends this work by performing sensitivity analysis on both algorithms (i.e., PSO and GA) to study the impact of involved parameters on generation of secure configuration.

V. Sensitivity Analysis

This section describes the methodology and algorithms of PSO and GA for sensitivity analysis.

A. Particle Swarm Optimization

A typical PSO algorithm consists of two parameters 1) velocity, and 2) fitness. These two parameters cooperate in order to locate the best possible configurations. In order to understand whether any of these parameters contribute more to the final best solution or if there is any interaction between them, we performed a sensitivity analysis on three scenarios, as follows:

1) Keeping particle fitness value range constant between [2,3) (*Algorithm 1*).
2) Keeping particle velocity value range constant as [0,1] (*Algorithm 2*).
3) Changing both velocity and particle fitness value range simultaneously (*Algorithm 3*).

The sensitivity analysis enables us to count how many particles with scores 2.0 the PSO algorithm would generate in every iteration. In following, we explain the algorithms in details.

1) Keep Fitness Value Constant: In Algorithm 1, PSO runs over the entire range of velocity from 0 to 8 taking two integer values ($v1$ and $v2$) at a time sequentially in every iteration of the `For` loop while keeping the fitness value range constant between [2, 3). In line 5, variable `particle_fitness` is assigned a list of 100 (`pop_size`) initial random `pbest` float values between 2 and 3. In line 6 `particle_vel` is assigned a list of 100 initial random velocity integer values between $v1$ and $v2$ inclusive, where $v1$ and $v2$ keeps changing. These two lists are then passed as an argument to the PSO algorithm which runs the PSO algorithm on these given initial lists and returns the count. The implementation of PSO algorithm called in line 7 is described in detail in paper [6].

Algorithm 1 PSO on Constant Fitness Value Range.

```
1: procedure CONSTANT_FITNESS
2:     particle_velocity_range = [0, 1, 2, 3, 4, 5, 6, 7, 8]
3:     pop_size = 100
4:     for v1,v2 in particle_velocity_range do
5:         particle_fitness = random.uniform(2, 3, pop_size)
6:         particle_vel =
7:             random.randint(v1, v2, (pop_size))
8:         cf = PSO(particle_fitness, particle_vel)
9:     end for
10: end procedure
```

2) Keep Velocity Value Constant: Algorithm 2 lists an algorithm similar to Algorithm 1 except that this time the velocity remains constant; whereas, the fitness values change systematically.

3) Change Fitness and Velocity Values: Furthermore, Algorithm 3 lists a scenario in which both parameters (i.e., fitness and velocity) are changing systematically.

B. Genetic Algorithm

Similar to the PSO's implementation, the GA part is developed in Python version 3.6. The GA outputs a pool of secure vulnerability pattern, where a pattern represents a chromosome in GA. The values it takes act as genes and the fitness function is dictated by the CVSS scores. The algorithm takes input, the best score, which is set 2.0 (i.e., the best and more secure

[2]https://nvd.nist.gov/

Algorithm 2 PSO on Constant Velocity Value Range.

procedure CONSTANT_VELOCITY
  $particle\_fitness\_range = [2, 3, 4, 5, 6, 7, 8, 9, 10]$
  $pop\_size = 100$
  for $f1$,$f2$ in $particle\_fitness\_range$ **do**
    $particle\_fitness = random.uniform(f1, f2, pop\_size)$
    $particle\_vel$ = $random.randint(0, 1, (pop\_size))$
    $cv = PSO(particle\_fitness, particle\_vel))$
  end for
end procedure

Algorithm 3 PSO on Changing Fitness and Velocity Value Ranges.

procedure CONSTANT_VELOCITY
  $particle\_fitness\_range = [2, 3, 4, 5, 6, 7, 8, 9, 10]$
  $particle\_velocity\_range = [0, 1, 2, 3, 4, 5, 6, 7, 8]$
  $pop\_size = 100$
  $fit\_vel = dict(zip(particle\_fitness\_range, particle\_velocity\_range))$
  for f,v in fit_vel **do** ▷ f & v each is a tuple taking 2 values at a time
    $pbest = random.uniform(f[0], f[1], pop\_size)$
    $particle\_vel = random.randint(v[0], v[1], (pop\_size))$
    $cfv = PSO(particle\_fitness, particle\_vel)$
  end for
end procedure

fitness score), number of iteration/generation = 50, mutation rate, and population size.

Algorithm 4 shows the pseudocode of how GA works. In-depth implementation of GA can found in paper [6]. We carried out sensitivity analysis on GA to see the effect on the number of secure (with $score = 2.0$) patterns generated by:

1) varying values of mutation rate from 0.2 to 1.
2) varying values of population size from 100 to 500 with an interval of 100.

We kept rest of the parameters constant. The next section discusses the results derived from sensitivity analysis.

VI. Results of Sensitivity Analysis

This section presents the results of sensitivity analysis conducted on model parameters of both Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) and discusses whether there is any changes in performance when some of the key parameters are treated as constants.

A. PSO Results

The PSO algorithm involves two parameters: 1) velocity, and 2) fitness. In this analysis, first we fix fitness values, and then observe the performance of the model (i.e., number of secure configuration generated) when changing velocity.

Algorithm 4 Pseudocode for GA.

procedure GA(best_score=2.0, gen=50, mut_rate, pop_size)
  Initialize g = 1
  Initialize pool of pop_size patterns
  while g!=50 **do** Selection based on score
    Crossover between chosen parent patterns
    Mutate child when $random.random > mut\_rate$
    g=g+1
  end while
  return pool of secure patterns
end procedure

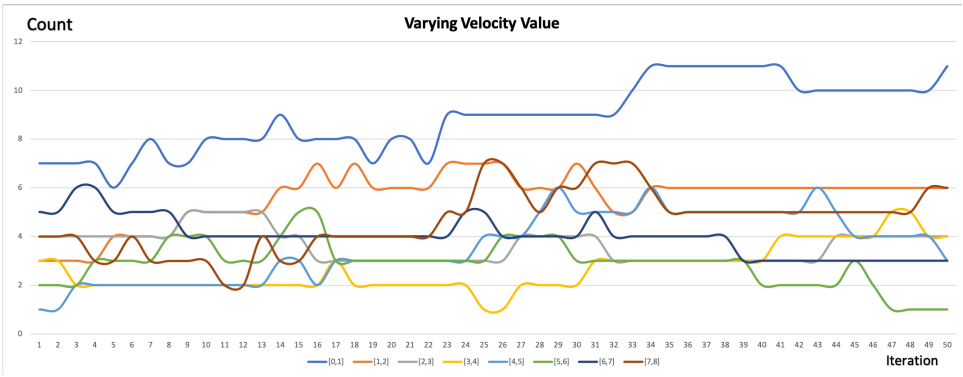
Second, the velocity is fixed and the performance is captured for varying fitness values. Finally, we repeat the process for varying both velocity and fitness values. The results evaluated in this section is for a single run of PSO. That means, the PSO script was run only once for every scenario described in the previous section. To be consistent with GA, every run of PSO is repeated through 50 iterations.

1) Varying Velocity Values: Figure 6a shows a scatter line plot for varying velocity value range in which count of particles/cvss vector strings with score 2.0 (y-axis) is plotted against every iteration (x-axis) for 50 iterations. Each coloured line plot represents different velocity value ranges PSO took as an input along with constant fitness range (i.e., $[2, 3)$).

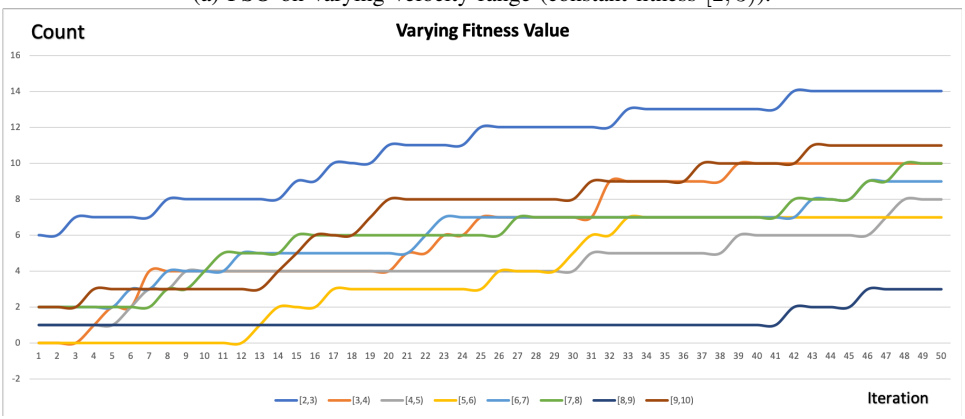
From the plot, we observe that PSO generated 11 best particles with velocity value range $[0, 1]$ and fitness range set to constant in the $[2, 3)$. The constant trend of the number of generated instances over 50 iterations for all cases indicate that the velocity parameter does not interact with fitness values and thus the variable is independently optimized itself. In other words, there is no observed interconnection between these two parameters.

On the other hand, the number of instances of best particles changes while the range of velocity values change. The maximum number of instances of best particles is achieved for $[0, 1]$ with 11 number of instances; whereas, the minimum number of instances of best particles is obtained for the velocity range $[5, 6]$ with only one instance produced (the 50-th iteration). This may indicate that for certain ranges of velocity, there is no improvement on performance even with the increasing number of iterations.

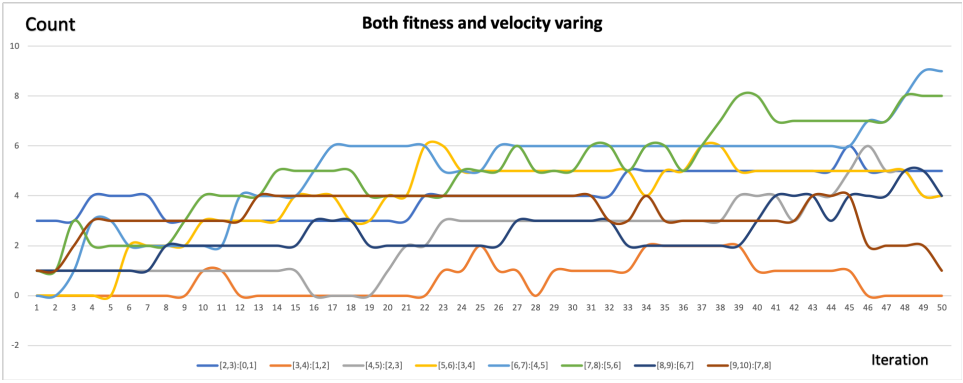
2) Varying Fitness Values: Figure 6b plots the results for the case when the fitness values are changed while the velocity value is kept in the constant range of $[0, 1]$. The maximum number of instances of the best vulnerable vectors is achieved for the fitness values in the range of $[2, 3)$ with 14 instances on the 50-th iterations. Whereas, the minimum number of instances (i.e., 3) is produced when the fitness value is in the range of $[8, 9)$. Unlike the previous scenario in which the velocity was changing and the number of produced instances was constant, in this case, the number of instances produced by the PSO algorithm is increasing over the iterations. A comparison of Figures 6a and 6b indicates that the fitness



(a) PSO on varying velocity range (constant fitness $[2, 3)$).



(b) PSO on varying fitness range (constant velocity $[0, 1]$).



(c) PSO on both varying fitness and velocity range

Fig. 6: Sensitivity analysis of PSO.

parameters positively contribute more to the PSO algorithm in producing best particles than the velocity parameter.

3) Varying Velocity and Fitness Values: Furthermore, in order to capture any possible interactions between velocity and fitness, we conducted a sensitivity analysis where both parameters are changing. Figure 6c illustrates the scenario where both velocity and fitness parameters are changing simultaneously. Each line plot is represented by `[f1,f2):[v1,v2]` where f_i and v_i represent the raneg of fitness and velocity, respectively. As figure demonstrates, PSO generated maximum of 9 best particles in its 50-th iteration for the input range $[6, 7) : [4, 5]$ where fitness range is $[6, 7)$ and velocity range is $[4, 5]$ with nine best particles produced. The trends captured in Figure 6c does not strongly indicates the existence of any interactions between these two parameters. However, it is important to note that our observation is limited to a few cases, as highlighted in the legend of Figure 6c. Some other combinations of velocity and fitness values may exhibit different behaviors.

B. GA Results

Similar to PSO, two parameters are involved in genetic algorithms (GAs): 1) mutation rate, and 2) population size. Similar to the sensitively analysis performed for PSA, we fix one parameter at a time and change the values of the other parameter and capture the changes occurred by this controlled

experimentation. Accordingly, we ran our GA script for 50 iterations.

1) Varying Mutation Rates: Unlike PSO, the sensitivity analysis performed on varying mutation rates while population size is fixed did not exhibit any clear pattern regarding improvement or effects of mutation rate on the number of secure configuration generated. Figure 7a demonstrates the results of sensitivity analysis. As it is apparent from the figure, there is no clear effects on the performance of the GA algorithm when mutation rates are systematically changed across different ranges of secure configuration. As there was no major pattern observed on the count of secure configurations generated on changing mutation rates for 0.2 to 1.0, we did not explore any other ranges for the rates.

2) Varying Population Sizes: Similarly, no clear changes were observed for the cases when the population size is varied. Figure 7b demonstrates the trends of changes observed in the performance of the GA algorithm (i.e., number of secure configuration generated) for varying population size. The results of sensitivity analysis performed on GA indicates that none of the parameters are influential in generating a better number and instances of secure configurations.

VII. Conclusion and Future Work

Cyber-Physical Systems are emerging technology and platforms in modern ubiquitous computing. The deployment of these systems has become more prevalent in recent years. Moving Target Defense (MTD) is an emerging technology in cyber defense. Enabling MTD allows modification of the underlying systems such that the surface exposed to cyber attacks are rapidly changing yet revealing minimal information to adversaries.

From theoretical standing point of view, it is feasible to change the configuration and settings of the underlying system continuously and thus implement an operation platform for deploying MTD systems. However, from practical point of view, it is also important that each newly generated configuration is as secure as the previous ones with minimal attack surface exposed to adversaries and fewer known vulnerabilities.

In our earlier work , we [6], [7] demonstrated how evolutionary algorithms such as Genetic Algorithms and Particle Swarm Optimizations can be useful in generating a set of secure configurations meeting required security expectations. In this paper, we presented a more in-depth analysis of showing the role of the parameters involved in these evolutionary algorithms. More specifically, we studied the importance of mutation rate and population size in Genetic Algorithms as well as the significance of velocity and fitness values in Particle Swarm Optimizations through sensitivity analysis.

The results of sensitivity analysis we conducted indicate that the GA's parameters (mutation rate and population size) are less influential on the performance of the algorithm where the performance is measured in terms of number of secure instances and configurations generated by GA. On the other hand, the parameters involved in Particle Swarm Optimizations algorithms are more influential in generating more secure configurations. In other words, changing the values of velocity and fitness values can affect the number of instances generated by the algorithm and thus should be taken into account when designing such evolutionary-based MTD platforms. Furthermore, our results show that employing Particle Swarm Optimization algorithms might be a better choice compared to Genetic Algorithms because of the influence of changing its parameters on improving the performance.

There exist some other research questions that need to be tackled in future. A critical research problem is the exploration of some other AI-based search problems in generating set of secure configurations. In particular, logic-based approaches such as Answer Set Programming (ASP) [16] can be useful for addressing the problem discussed in this paper. ASP solvers are able to generate answer sets, where each set is a secure configuration of the underlying system. The prominent problem is the logical representation of the secure configuration problem in terms of Answer Set Programming. In addition to logic-based approaches, it is also possible to formulate such problem with machine learning modules such as reinforcement learning in order to generate a set of secure configurations [3].

Acknowledgment

This research work is supported by National Science Foundation (NSF) under Grant No: 1821560.

References

[1] Keivan Borna and Vahid Haji Hashemi. An improved genetic algorithm with a local optimization strategy and an extra mutation level for solving traveling salesman problem. *International Journal of Computer Science, Engineering and Information Technology*, 4(4):47–53, Aug 2014.

[2] J. Carr. An introduction to genetic algorithms. 2014.

[3] Moitrayee Chatterjee and Akbar Siami Namin. Detecting phishing websites through deep reinforcement learning. In *IEEE COMPSAC*, pages 227–232, 2019.

[4] M. Crouse and E. W. Fulp. A moving target environment for computer configurations using genetic algorithms. In *2011 4th Symposium on Configuration Analytics and Automation (SAFECONFIG)*, pages 1–7, Oct 2011.

[5] Michael B. Crouse, Errin W. Fulp, and Daniel A. Cañas. Improving the diversity defense of genetic algorithm-based moving target approaches. 2012.

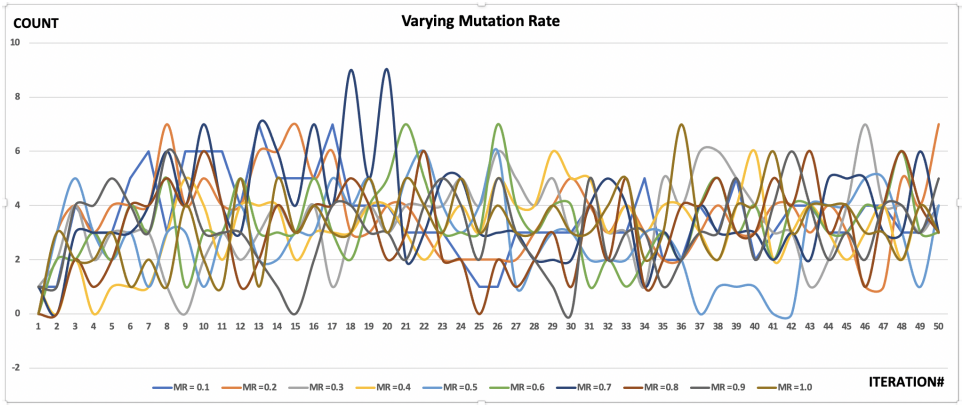
[6] S. Dass and A. Siami Namin. Evolutionary algorithms for vulnerability coverage. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1795–1801, 2020.

[7] Shuvalaxmi Dass and Akbar Siami Namin. Vulnerability coverage for adequacy security testing. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 540–543, New York, NY, USA, 2020. Association for Computing Machinery.

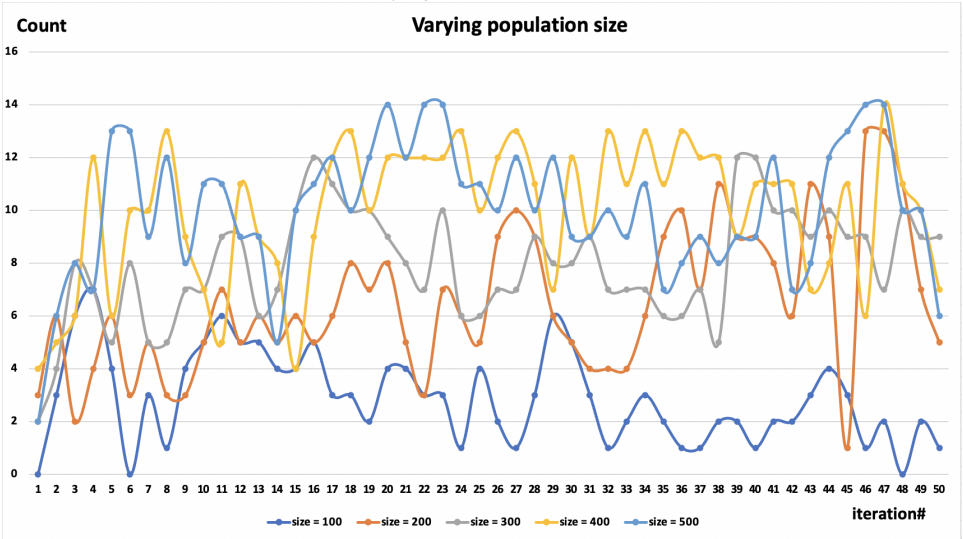
[8] Stefan Hoops, Raquel Hontecillas, Vida Abedi, Andrew Leber, Casandra Philipson, Adria Carbo, and Josep Bassaganya-Riera. Chapter 5 - ordinary differential equations (odes) based modeling. In Josep Bassaganya-Riera, editor, *Computational Immunology*, pages 63 – 78. Academic Press, 2016.

[9] David J. John, Robert W. Smith, William H. Turkett, Daniel A. Cañas, and Errin W. Fulp. Evolutionary based moving target cyber defense. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1261–1268, New York, NY, USA, 2014. ACM.

[10] V. Long Do, L. Fillatre, and I. Nikiforov. Sensitivity analysis of the sequential test for detecting cyber-physical attacks. In *2015 23rd European Signal Processing Conference (EUSIPCO)*, pages 2261–2265, 2015.



(a) Varying mutation rate in GA



(b) Varying population size in GA

Fig. 7: Sensitivity analysis of GA.

[11] Yuriy Zacchia Lun, Alessandro D'Innocenzo, Francesco Smarra, Ivano Malavolta, and Maria Domenica Di Benedetto. State of the art of cyber-physical systems security: An automatic control perspective. *Journal of Systems and Software*, 149:174–216, 2019.

[12] Mei-Ping Song and Guo-Chang Gu. Research on particle swarm optimization: a review. In *International Conference on Machine Learning and Cybernetics*, 2004.

[13] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 2013.

[14] Caroline Anne Odell. Using genetic algorithms to detect security related software parameter chains. Master's thesis, Wake Forest University Graduate School Of Arts And Sciences, Winston-Salem, North Carolina, 5 2016.

[15] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1):33–57, 2007.

[16] Sara Sartoli and Akbar Siami Namin. Modeling adaptive access control policies using answer set programming. *J. Inf. Secur. Appl.*, 44:49–63, 2019.

[17] Irfan Syamsuddin. Multicriteria evaluation and sensitivity analysis on information security. *International Journal of Computer Applications*, 69(24):22–25, May 2013.

[18] Xiaohui Hu, Yuhui Shi, and R. Eberhart. Recent advances in particle swarm. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 1, pages 90–97 Vol.1, June 2004.

[19] Jianjun Zheng and Akbar Siami Namin. A markov decision process to determine optimal policies in moving target. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2321–2323, 2018.

[20] Jianjun Zheng and Akbar Siami Namin. A survey on the moving target defense strategies: An architectural perspective. *J. Comput. Sci. Technol.*, 34(1):207–233, 2019.