Using Genetic Algorithms for Deadline-Constrained Monitor Selection in Dynamic Computer Networks

Robin Mueller-Bady
Frankfurt University of Applied Sciences
Nibelungenplatz 1
Frankfurt am Main, Germany
mueller-bady@fb2.fra-uas.de

Ruediger Gad
Frankfurt University of Applied Sciences
Nibelungenplatz 1
Frankfurt am Main, Germany
rgad@fb2.fra-uas.de

Martin Kappes
Frankfurt University of Applied Sciences
Nibelungenplatz 1
Frankfurt am Main, Germany
kappes@fb2.fra-uas.de

Inmaculada Medina-Bulo
Universidad de Cádiz
C/Chile 1
CP 11002 Cádiz, Spain
inmaculada.medina@uca.es

ABSTRACT

In this paper we address the problem of selecting a minimal number of optimally positioned monitors for capturing network traffic in dynamic computer network environments. Requirements of computer network monitoring change frequently, e.g., due to indicators of an ongoing attack, which requires a continuous optimization and adaptation of the monitoring state. The monitor selection problem can be mapped to the vertex cover problem which NP-complete. Therefore, we propose a genetic algorithm (GA) as optimization heuristic for obtaining an appropriate solution in adequate time.

In case of incidents in the monitored computer network, it is necessary to adapt the monitoring strategy swiftly, reliably and frequently. This is why we constrain the heuristic using a hard deadline for the optimization process indicated by the number of processed generations of the GA. Necessary parameters for the GA were studied in order to optimize them with respect to the given time constraint.

The GA was applied to different computer network graph models generated using the Barabasi-Albert model with 30 and 100 vertices and the "National Research and Education Network" (NREN) Europe. We show that the proposed GA provides reliable and valuable results in an adequate time.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network Monitoring; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic Methods*

GECCO '15 Companion, July 11 - 15, 2015, Madrid, Spain
 ISBN 978-1-4503-3488-4/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2739482.2768430

Keywords

Genetic Algorithm, Computer Network, Monitor Selection, Network Monitoring

1. INTRODUCTION

Over the past decades, computer networks became an indispensable part of todays business-, research- and personal sector. Both, quality and quantity of computer networks increased due to the higher amount and different types of devices in communication networks, e.g., mobile phones, tablet computers, or smart home devices[18]. At the same time, those devices require a higher bandwidth to fulfill their requirements, e.g., Voice over IP (VoIP) calls, video, or music streaming.

The development of modern computer networks also increases security concerns related to those communication networks. Data theft, fraud, and (denial of service) attacks become more frequent as they also imply higher significance on business- or safety-critical systems[17]. It is necessary to address these security related problems with high attention.

The first step of the successful implementation of countermeasures against attacks is the gathering of information about the current underlying computer network. This is usually done using passive network monitoring techniques, i.e., sniffing of network traffic. To accomplish this task efficiently, it is necessary to identify optimal positions in a computer network to install monitors, which can either be specific hardware devices or software on existing networking infrastructure, e.g., switches or routers. For optimizing efficiency, it is also important to identify the optimal number of monitors. A large quantity of monitors leads to high costs for implementation and maintenance. It has also a higher negative impact on the computer networks performance due to the monitoring process and exchange of information between the monitors or a central analysis server. However, too few monitor positions will lead to loss of information of the target network and therefore to a higher security risk. An adequate amount of and positions for monitors in the network must be found such that they cover all traffic that passes on adjacent links.

In our work, the best position for a monitor in a network is determined based on a snapshot of the current situation. A shift of the traffic in the network, e.g., based on an ongoing attack, might change the situation for the monitors drastically. It is necessary to be able to react rapidly on changing requirements of the current situation of a network in order to preserve a high security level. The upcoming technology of software-defined networking (SDN) is one of the key technologies to enable this rapid change of the configuration of a computer network, e.g., its monitors. This technology leverages the advantages of a continuous optimization of the monitoring configuration of dynamic computer networks.

In this paper we use genetic algorithms (GAs) for the optimization process. Humans often tend to design a systems, devices, machines, or structures by the means of aesthetics, symmetry and order. As Keane and Brown show, better solutions for a given problem might not always be symmetric or aesthetic: they optimize a satellite beam regarding its stability in space using evolutionary computing methods by a factor of 200[20]. There are several advantages of using GAs for this type of optimization problem[15, 5]. GAs are known to perform well on computational hard problems with constraints and a high number of input parameters. Furthermore, compared to other greedy local search techniques, e.g., hill-climbing, they are flexible regarding the search space, especially in case the search space is multimodal, i.e., it has many local optima which may trap other search methods. Finally, in our specific case, genetic algorithms continuously provide results after each generation, i.e., the GA might be disrupted during the optimization process still providing the currently best result. This might helpful in cases where an urgent change of the monitoring state of the current situation is required.

This paper addresses those issues and shows, how to solve them using genetic algorithms.

2. RELATED WORK

The problem of finding an optimal amount of monitors and their optimal positions can be mapped to the NP-complete minimum vertex cover optimization problem[26]. Thus, assuming $P \neq NP$, there is no algorithm to compute an optimal solution to a given instance of the problem in acceptable time. For this reason, we apply genetic algorithms as probabilistic optimization heuristic.

There are many systems which provide an architecture for monitoring different types of information in a network, e.g., available bandwidth[1], traffic information on flow level[23], or multi-tenant cloud infrastructures[22]. However, all of them are facing the problem of finding an optimal coverage for their network in order to minimize the necessary resources for the monitoring process. Chaudet et al.[6] propose an integer-programming method to place a fixed number of monitors on optimal positions in the network. Chen et al.[7] study the placement of intrusion detection system sensors in a network for surveillance purposes. Both approaches solve design problems, where the genetic algorithm is used once before the implementation of the final solution in the target network. Our solution is designed in order to continuously optimize the existing dynamic computer network having a time constraint. For our problem, the NP-complete integer programming paradigm does not meet the requirements of having a flexible and fast solution for the optimization problem.

Applying evolutionary computing to graph-based problems from many different disciplines is common[13]. Syarif et al.[24] propose a method to optimize a multi-stage supply chain network (SCN) regarding their cost using a spanning-tree based genetic algorithm. Altiparmak et al.[3] developed another genetic algorithm for the optimization of the same SCN but based on multiple objectives: transportation cost, delivery time, and optimization of the capacity utilization balance of storage centers. Other graph-based problems are researched by Gen et al.[12, 13, 14], developing genetic algorithms for several graph-based problems, e.g., the minimum spanning-tree problem, design of local area networks, fixed-charge transportation problem, or the centralized network design problem.

Regarding the specific problem of selecting an optimal number of monitors in a given network graph, Zhu et al.[25] provide an overview of existing monitor selection methods in computer networks. They are based on information which finally helps to determine the quality of the given network with respect to quality of service (QoS) attributes, e.g., bandwidth and packet drop rate.

Our approach can also be used for distributed monitoring of network traffic, as shown by Gad et al.[11]. In this work, a method for distributing network traffic capturing is introduced based on simple classifications of packets based on packet header field information. They also propose an approach to dynamically monitor network traffic on certain points in a network in a self-adaptive and cooperative manner[10]. Both applications can benefit from a continuous selection of optimal monitoring positions. However, all those proposed methods do not fulfill the requirement of being flexible enough to react on recent changes in the network. Therefore, our solution can be used to bridge that gap.

3. PROBLEM

As model for reflecting the computer network, we assume an undirected simple graph with a fixed topology $G = (V, E, w)$ with V being the set of vertices, $E \subseteq V^2$ being the set of edges which are symmetric tuples of vertices, and $w : E \to \mathbb{N}^+$ being a non-negative weighting function for the edges representing the priority of the edge to be monitored. For the remainder of this paper, we follow the graph notation and definitions of[8]. We study the problem of finding the optimal number and positions of monitors in a computer network, such that the number of monitors is minimal and the positions are chosen in order to cover each edge. Having a graph $g \in G$ whose edge weightings are all 1, this problem can mapped to the vertex cover problem, which is defined as follows: Let $G = (V, E)$ be an undirected simple graph, a vertex cover is a subset of vertices $V' \subseteq V$ such that for each edge $(v_i, v_j) \in E$, either $v_i \in V'$ or $v_j \in V'$, or both. The minimum vertex cover is a vertex cover where $k = |V'|$ is minimal for all possible vertex covers of G[19].

4. GENETIC ALGORITHM

We propose a general generational genetic algorithm as search strategy. For the experiments, we pick a population size of 200 for all GA runs, as this provides an adequate amount of genetic diversity for the given problem. The initial population is randomly generated. As the number of generations is our key indicator for the runtime of the GA, we will vary it over four steps: 50, 100, 500, and $1,000$. In

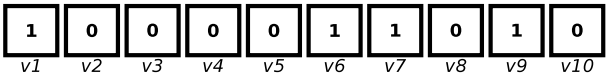

Figure 1: Genotype of an individual in the genetic algorithm indicating the selected monitors

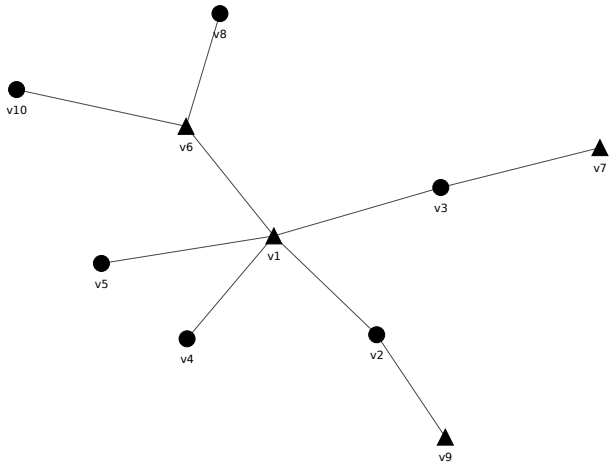

Figure 2: Phenotype of an individual in the genetic algorithm indicating the selected monitors

the following, the individual components of the GA will be discussed in more detail.

4.1 Representation

While there exist different approaches for representations of graph based problems in GAs[12], we follow a vertex-based approach for the proposed GA. Let $G = (V, E, w)$ be a graph. We use a binary vertex-based genotype encoding string, $\vec{x} \in \{0,1\}^{|V|}$, such that each gene, $(x_1, ..., x_n) \in \vec{x}$, is representing one vertex in the graph:

$$x_i = \begin{cases} 1, & if\ v_i\ selected\ as\ monitor \\ 0, & otherwise \end{cases} \tag{1}$$

This results in a binary string representing one individual from the result space. An example for the genotype of such an individual is shown in Figure 1. See Figure 2 for the phenotypical graph representation of this individual. The monitored vertices are represented by triangles, whereas vertices not acting as monitors have a circle representation.

4.2 Evaluation

The fitness for the individuals is evaluated using the network graph model as defined before in conjunction with the genotype of the individual indicating the positions of the monitors. As this problem is defined as minimization problem, a lower fitness indicates a better result. Calculation of the fitness is done as follows: Let $G = (V, E, w)$ be a graph and $(x_1, \ldots, x_n) \in \vec{x}$ a solution instance of the problem. First we calculate the number of selected monitors

$$monitors(\vec{x}) = \sum_{i=1}^{|V|} x_i \tag{2}$$

where $\vec{x}$ is an individuals genotype and x_i gene at position i, respectively. In case of an individual with an infeasible solution, i.e., a solution that does not cover all edges, we introduce a penalty which penalizes the solution with the double of its weighting, such that

$$penalty(\vec{x}) = \sum_{(v_i,v_j)\in E} c \cdot w(v_i, v_j) \tag{3}$$

$$\text{where } \{(v_i, v_j) \in E \mid x_i + x_j = 0\} \tag{4}$$

and c is a constant penalty factor. We use a penalty factor $c = 2$ for our experiments. In this case, having an uncovered edge is worse than having too many monitors. With $c < 2$, the fitness distance of solutions having uncovered edges compared to solutions with a higher number of monitors but less uncovered edges decreases. Having $c > 2$, this distance increases but the GA becomes greedier. In our case, both are undesirable behaviors. Additionally, with this factor we ensure that the minimal vertex cover is always the solution with the best fitness value.

Finally, the total fitness for each individual is calculated as follows:

$$fitness(\vec{x}) = monitors(\vec{x}) + penalty(\vec{x}) \tag{5}$$

which is in the range of

$$\left[1, \left(|V| + \sum_{(v_i,v_j)\in E} c \cdot w\left(v_i, v_j\right)\right)\right] \tag{6}$$

assuming a graph with $V \neq \emptyset$.

4.3 Genetic Operators

In the following, the used genetic operators and their parameters for the GA will be described. For the given problem, we follow the steps of the simple genetic algorithm as described by Baeck et al.[4]: First, the population is evaluated using the defined objective (fitness) function. For a given number of generations, the existing population creates their offspring using selection, crossover and mutation operations. The new population is evaluated and selected according to the selection algorithm. This process repeats until a certain termination condition is met. In case of the proposed GA, the termination condition is the number of processed generations.

4.3.1 Recombination

The recombination operation is used in order to explore a new area of the search space with the help of two parent individuals. We researched different recombination operators, including simple 1-point crossover, 2-point crossover, and partially mapped crossover (PMX)[4, 16]. The 1-point crossover operation randomly picks 1 position in the genotype string and swaps all genes beyond. The 2-point crossover operation works similar except that two points are chosen which mark start and end position of the gene swapping interval. PMX is usually used for permutation based problem representations. It works by randomly choosing two crossover points and exchange position of genes in both parents without changing the set of elements in the individual. This crossover operation only changes the order of elements in the genotype string according to their parents. We considered this operation for this genetic algorithm as we experienced beneficial results in a later stage of the GA run when primarily placement of monitors is important over the reduction of monitoring points. See Figure 3 for an evaluation of the crossover operations for an average of 100 GA

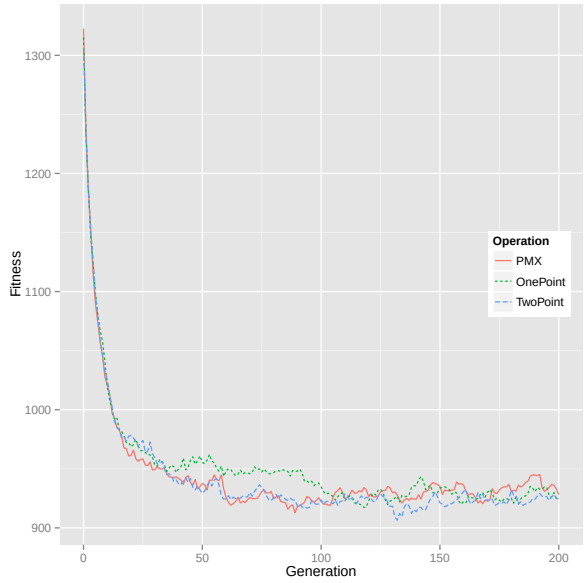

Figure 3: Evaluation of the crossover operations

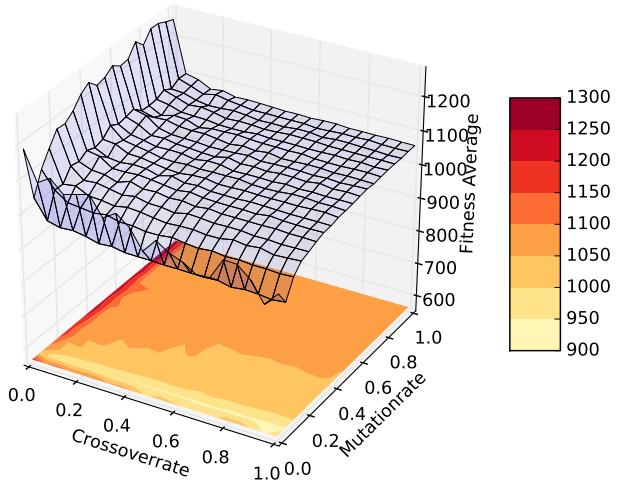

Figure 4: Evaluation of the crossover probability parameter p_{cx}

runs of the given problem. As shown in the figure, there is just a negligible difference between the researched crossover methods for the underlying problem. However, the PMX operation has slight advantages as it converges earlier to a good result. Thus, we use PMX as the preferred crossover operation.

We studied the different parameter options for the crossover probability, as can be seen in Figure 4. In this figure, we show the mutation rate, p_{ind} in relation to the crossover rate p_{cx} and the fitness average of the GA run. A population size of 200, the PMX crossover operation, shuffle mutation and tournament selection with tournament size 20 and 400 generations is used. The step size of the crossover and mutation rate is 0.05 per operation. A low fitness average value of a certain p_{ind}/p_{cx} combination determines a high suitability for the GA.

As can be seen in the figure, a value of $p_{cx} = 1.0$ is beneficial for our GA which means that each tuple of parents of the population will produce exactly two children.

4.3.2 Mutation

For the exploration of the local search space of the current solution, a shuffle operation is used as the mutation operator. This operation is used in order to avoid premature convergence but also introduce slight changes to the genotype of the individual. We studied different mutation operations for this problem, especially the common bit flip and shuffle mutation operation. The bit flip operation introduces a change in the genotype by flipping an allel with a certain probability, whereas the shuffle mutation picks two different alleles and changes their position. As can be seen in Figure 5, the shuffle operation performed better than the bit flip operation for an average of 100 runs of the proposed GA. For this reason we use the shuffle mutation as mutation operation. We distinguish between two different types of mutation probabilities: the mutation probability for the individual components (alleles) (p_{ind}) and the general mutation probability (p_{mut}). Probability value p_{mut} is used in

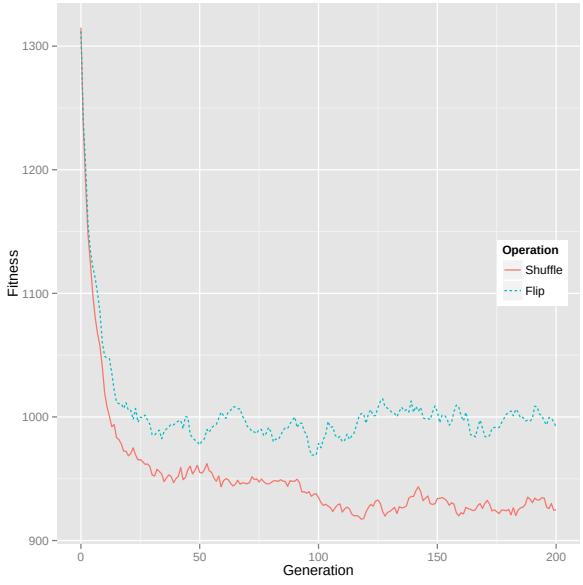

Figure 5: Evaluation of the mutation operations

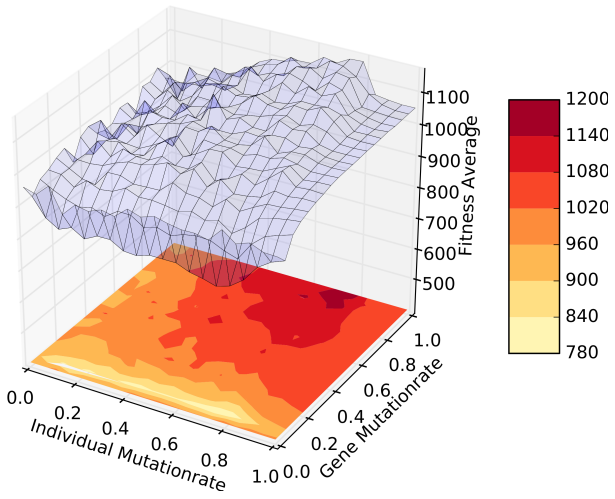

Figure 6: Evaluation of the p_{ind} and p_{mut} parameters

order to distinguish whether an individual is considered for mutation at all, whereas p_{ind} defines the probability for altering single alleles in case an individual is considered for the mutation.

We studied the different probabilities for the underlying problem, as can be seen in Figure 6. In this figure we compared the fitness values of 400 generations of the individual and the allel mutation probability with a step size of 0.05 for each value from 0.0 to 1.0. The following values yield the best results in the evaluation of possible parameters: $p_{mut} = 0.8$ and $p_{ind} = 0.05$.

4.3.3 *Selection*

Selection was implemented using a tournament selection[4]. Within this selection type, a random choice of individuals from the population is used to participate in a tournament. The best individual of the tournament gets selected. This procedure is repeated until the number of necessary selections is reached. For the tournament selection, the parameter for the tournament size is chosen as one-tenth of the population size. We avoid using elitist approaches due to the huge multimodal search space.

5. EXPERIMENTS

For our experiments, we use three different network graph models. The first two graphs are generated following the Barabási-Albert model[2] generating scale-free networks having 30 and 100 vertices, respectively. The weighting for the edges is randomly generated using the interval $]0, 10]$ indicating the priority of the edge being monitored where 1 indicates a low and 10 a high priority. Both graphs are sparse as they have the minimum number of edges, $|V| - 1$.

The data for the third network is provided by the National Research and Education Network (NREN) Europe[21]. The model contains the data set of the European NREN interconnect model which forms the backend for the European research network. This network model consists of 1,157 vertices and 1,465 edges. We apply the same random weighting in the interval of $]0, 10]$ as for the other two graphs. However, this graph has, compared to the first two, a higher rate of interconnecting vertices and is therefore tighter.

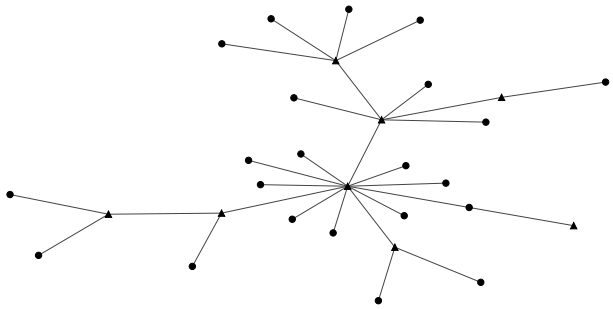

Figure 7: Graph $|V| = 30$

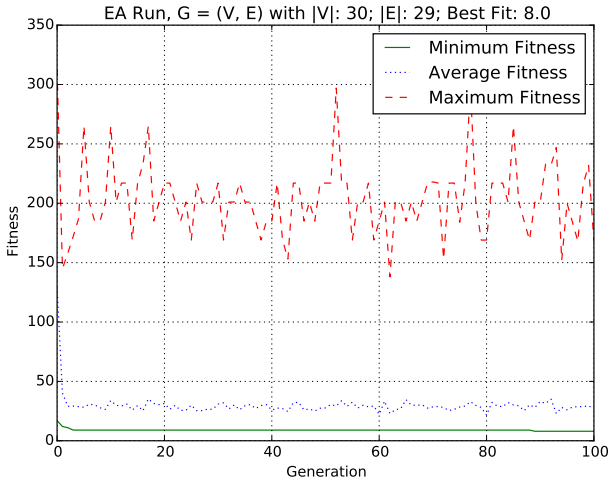

Figure 8: GA statistics for graph $|V| = 30$

Each result of the GA is averaged over 10 runs per generation and network model. The implementation of the genetic algorithm is done using python, deap, and igraph[9].

6. RESULTS

The results of the experiments are shown in Table 1.

For the graph with $|V| = 30$, the GA already converges early to a result close to the final one using the minimal number of generations. For all cases of generations of this network model, there is no uncovered edge within the graph. As there applies no penalty for uncovered edges, the fitness number of monitors is equal to the fitness. For all cases of different generation deadlines, the fitness is equal.

Figure 7 provides an illustration of the resulting graph with the highest fitness discovered during the run of the GA with 100 generations. The statistics for this run are shown in Figure 8, where the minimum, average, and maximum fitness are plotted in relation to the generations over the whole GA run. As can be seen, the GA converges in an early generation to a result close to the final result. In this case, the best result of the heuristic run appears first in generation ~ 90. The high variation of the maximum fitness indicates a high genetic variation during the search process, whereas the low variation of the average and minimum fitness shows that there exists a stable subpopulation near the current optimum.

For the graph with $|V| = 100$, the results in Table 1 show that all edges are covered for all generation deadlines. How-

Table 1: Results of the GA - using deadlines for introduced network graph models

Network Model	Generations	Covered Edges	Uncovered Edges	Monitors	Fitness
Graph $\|V\| = 30$	50	29	0	8	8
Graph $\|V\| = 30$	100	29	0	8	8
Graph $\|V\| = 30$	500	29	0	8	8
Graph $\|V\| = 30$	1000	29	0	8	8
Graph $\|V\| = 100$	50	99	0	32.1	32.1
Graph $\|V\| = 100$	100	99	0	30.6	30.6
Graph $\|V\| = 100$	500	99	0	29.8	29.8
Graph $\|V\| = 100$	1000	99	0	29.7	29.7
Graph "NREN"	50	1386.0	79.0	631.8	789.8
Graph "NREN"	100	1397.6	67.4	637.7	773.5
Graph "NREN"	500	1414.9	50.1	631.6	731.8
Graph "NREN"	1000	1419.5	45.5	632.5	723.5

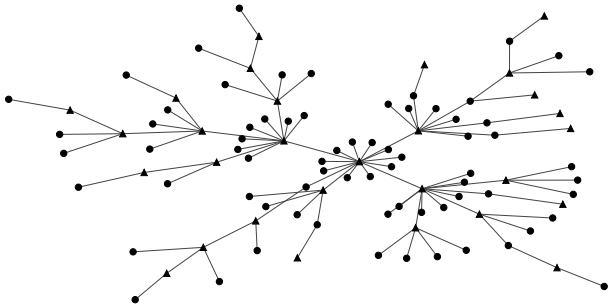

Figure 9: Graph $|V| = 100$

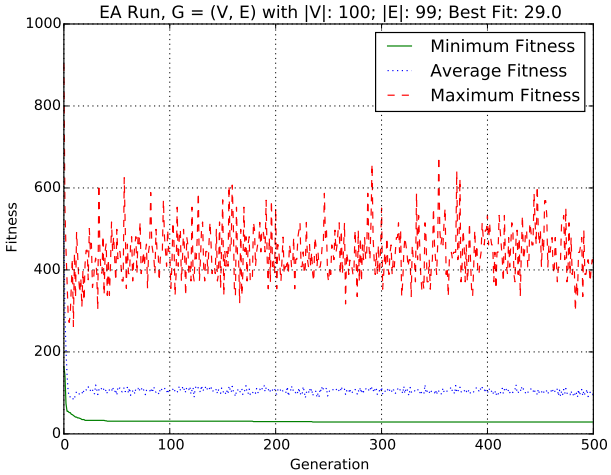

Figure 10: GA statistics for graph $|V| = 100$

ever, the number of monitors required to cover all edges improves noticeable. For the case of 50 generations, the average number of monitors is 32.1. The GA run with 100 generations improves this to an average number of monitors of 30.6, which is an improvement of 4.67%. This average improvement factor reduces with the number of generations, e.g., between 500 and 1,000 generation it is 0.34%, but it is shown that improvement is still possible. From the deadline of 50 generations to 1,000 generations, the total average improvement factor is 7.48%.

The statistics for one exemplary GA run in Figure 10 and the final graph in Figure 9 confirms this observation. In the beginning of the GA run, there is an improvement in the fitness level of the individuals until generation ~ 50. As for the first graph, the GA also converges in an early generation to a fitness value close to the final result. After that, only minor changes to the fitness are introduced. As described before, the high variation of the maximum fitness indicates a high genetic variation during the search process, whereas the low variation of the average and minimum fitness indicates the stable subpopulation.

For the NREN graph, the results show that a significantly higher amount of generations is necessary in order to produce satisfactory results. After the first 50 generations, the GA produces a result were 5.39% of the edges are not covered, this reduces to 4.60% after 100 generations, 3.42% after 500 generations and finally to 3.11% after $1,000$ generations. Simultaneously, the number of monitors remains stable throughout all generations including minor deviations, e.g. -0.93% from 50 to 100 generations or $+0.14\%$ from 500 to 1,000 generations.

The final graph in Figure 11 and the related statistics for this GA run in Figure 12 show noticeable improvements over the generations of the GA. The GA does still maintain a high genetic variation as indicated by the maximum fitness values, but minimum and average are constantly decreasing, even after generation 800. This run of the GA requires ~ 300 generations until it reaches a fitness level close to the final fitness. The GA converges to the final fitness in generation ~ 850, which shows that it is still worthwhile to use a higher amount of generations for the GA to access better results.

Indeed, we can still experience improvement by increasing the number of generations, which is shown in another experimental run of the GA for the NREN network model using 5,000 generations. The results are shown in Figure 13. This plot shows that the GA is still able to introduce improvements on the given network graph model, even after the defined deadline of 1,000 generations.

Table 2 shows the ratios of uncovered edges and monitors for the different graphs and number of generations. As can be seen, the coverage ratio for small and medium sized networks does not change as it already reaches its maximum in early generations. For the network graph model containing 30 vertices, even the number of monitors is constant for all runs of the GA. However, for the network graph model with

Table 2: Results of the GA - coverage and monitor ratio

	Graph 30		Graph 100		Graph NREN	
Generations	Uncovered	Monitors	Uncovered	Monitors	Uncovered	Monitors
50	0.00%	26.67%	0.00%	32.10%	5.39%	54.61%
100	0.00%	26.67%	0.00%	30.60%	4.60%	55.12%
500	0.00%	26.67%	0.00%	29.80%	3.42%	54.59%
1000	0.00%	26.67%	0.00%	29.70%	3.11%	54.67%

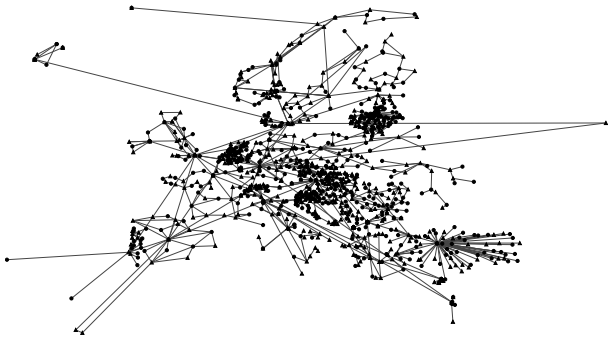

Figure 11: Graph "NREN"

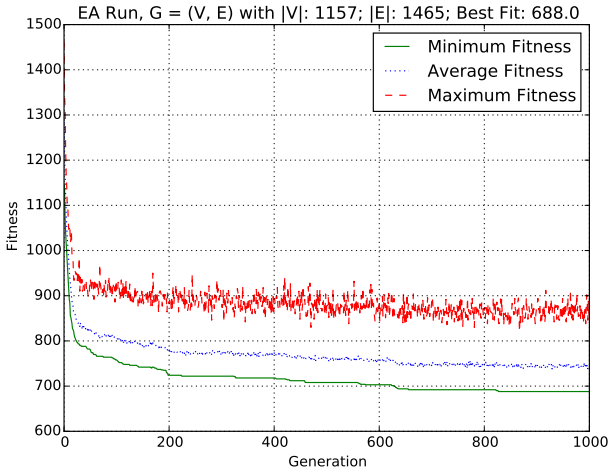

Figure 12: GA statistics for graph "NREN" after 1,000 generations

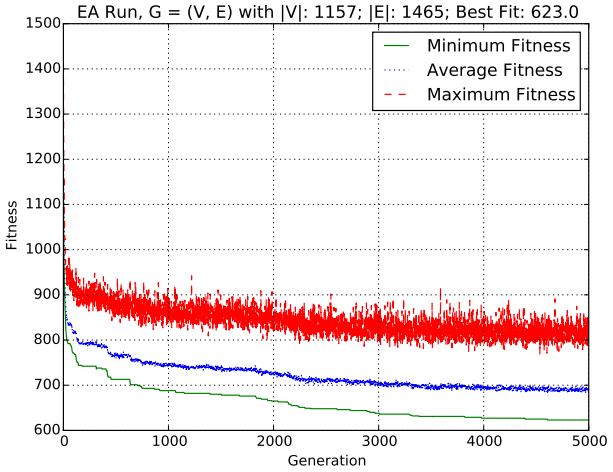

Figure 13: GA statistics for graph "NREN" after 5,000 generations

100 vertices the number of required monitors reduces during later generations while maintaining the same coverage.

For the more complex NREN network graph model, the situation is different. Here, a low number of uncovered edges is found in early generations but it is still decreasing during the run of the GA. While still increasing the coverage, the number of monitors remains constant as also described in the results before.

7. CONCLUSION

In this paper we developed a genetic algorithm for finding continuously rapid, reliable and adequate solutions for the problem of selecting monitors in a dynamic computer network. The GA was applied to two generated and a real network graph model with 30, 100, and 1157 vertices having a random weighting within $]0, 10]$ indicating the importance of a certain edge to be monitored.

As the results show, the proposed GA leads to good results for small to medium sized networks after just a few generations. For those networks, further computing might still lead to better results regarding the number of monitors while the ratio of covered versus uncovered edges remains constant. Having a more complex network, the amount of necessary generations for creating results with similar quality compared to the simple networks increases, but satisfying solutions are still produced in early generations.

On average, a higher number of generations leads to fewer number of monitors for, at least, the same amount of covered edges as shown before. This increases the coverage of the monitors which in the end leads to higher security and fewer cost of the monitoring process.

Our approach can be beneficial for usage in IT-Security related systems, e.g., traffic monitoring or intrusion detection systems. This approach can be used in dynamic systems which depend on fast reactions on rapidly changing requirements due to, e.g., attacks. Especially in dynamic computer networks such as virtual environments or software defined networks, where configuration can be automated, this approach can be very beneficial. As future work, the research continues increasing the network graph models size and the further development of the genetic algorithm using new operations, EA models and multiple (concurrent) objectives.

8. ACKNOWLEDGMENTS

This work was supported in the framework of Hessen ModellProjekte, financed with funds of the European Union (European Regional Development Fund - ERDF) and the State of Hessen in the context of the research project "Reactive network Optimization by Using SDN-Technology" (ROBUST) (HA project no. 123/14). Responsible for the content are the authors.

9. REFERENCES

[1] G. Aceto, A. Botta, A. Pescapé, and M. Darienzo. Unified architecture for network measurement: The case of available bandwidth. *Journal of Network and Computer Applications*, 35:1402–1414, 2012.

[2] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(January):48–94, 2002.

[3] F. Altiparmak, M. Gen, L. Lin, and T. Paksoy. A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & Industrial Engineering*, 51(1):196–215, 2006.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC Press, 1st edition, 2000.

[5] T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

[6] C. Chaudet, E. Fleury, I. G. Lassous, H. Rivano, M.-E. Voge, I. Guerin, I. D. Lyon, and E. Voge. Optimal positioning of active and passive monitoring devices. *CoNEXT 2005 - Proceedings of the 2005 ACM Conference on Emerging Network Experiment and Technology*, pages 71–82, 2005.

[7] H. Chen, J. a. Clark, S. a. Shaikh, H. Chivers, and P. Nobles. Optimising IDS sensor placement. In *ARES 2010 - 5th International Conference on Availability, Reliability, and Security*, pages 315–320, 2010.

[8] R. Diestel. *Graph Theory*. Springer-Verlag, Heidelberg, 4th editio edition, 2010.

[9] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagne. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*, 13:2171–2175, 2012.

[10] R. Gad, M. Kappes, and I. Medina-Bulo. Monitoring Traffic in Computer Networks with Dynamic Distributed Remote Packet Capturing. In *IEEE ICC 2015 - Next Generation Networking Symposium (ICC'15 (07) NGN)*, 2015.

[11] R. Gad, M. Kappes, R. Mueller-bady, and I. Medina-Bulo. Header Field Based Partitioning of Network Traffic for Distributed Packet Capturing and Processing. In *IEEE 28th International Conference on Advanced Information Networking and Applications (AINA)*, pages 866–874, 2014.

[12] M. Gen and R. Cheng. *Genetic algorithms and engineering optimization*, volume 7. 2000.

[13] M. Gen, R. Cheng, and S. S. Oren. Network design techniques using adapted genetic algorithms. *Advances in Engineering Software*, 32:731–744, 2001.

[14] M. Gen, A. Kumar, and J. R. Kim. Recent network design techniques using evolutionary algorithms. *International Journal of Production Economics*, 98:251–261, 2005.

[15] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[16] D. E. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, 1985.

[17] IBM Global Technology Services. IBM Security Services Cyber Security Intelligence Index. Technical report, 2013.

[18] International Telecommunication Union. International Telecom Union Annual Report 2013: Measuring the Information Society. Technical report, Geneva, Switzerland, 2013.

[19] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[20] A. Keane and S. Brown. The design of a satellite beam with enhanced vibration performance using genetic algorithm techniques. *The Journal of the Acoustical Society of America*, 99:2599–2603, 1996.

[21] S. Knight, N. Falkner, H. X. Nguyen, P. Tune, and M. Roughan. I can see for miles: Re-visualizing the internet. *IEEE Network*, 26(December):26–32, 2012.

[22] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Generation Computer Systems*, 29(8):2041–2056, 2013.

[23] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. CSAMP: A System for Network-wide Flow Monitoring. *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 233–246, 2008.

[24] A. Syarif, Y. Yun, and M. Gen. Study on multi-stage logistic chain network: A spanning tree-based genetic algorithm approach. *Computers and Industrial Engineering*, 43:299–314, 2002.

[25] N. Zhu, J. Zuo, Y. Zhou, and W. Wang. Overview of Monitor Selection in Computer Networks. In Y. Yuan, X. Wu, and Y. Lu, editors, *Trustworthy Computing and Services*, pages 52–59. Springer Berlin Heidelberg, 2013.

[26] X. Zhuo and X. Cao. Research based on the specific optimizing strategy for network flow monitoring. In *3rd IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, pages 294–298, 2012.