Improving Stress Search Based Testing using a Hybrid Metaheuristic Approach

Francisco Nauber Bernardo Gois
Serviço Federal de
Processamento de Dados
Avenida Pontes Vieria ,832
Fortaleza, Ceará, Brazil
francisco.gois@serpro.gov.br

Pedro Porfírio Muniz de Farias
Universidade de Fortaleza
Av. Washington Soares, 1321
Fortaleza, Ceará, Brazil
porfirio@unifor.br

André Luís Vasconcelos Coelho
Universidade de Fortaleza
Av. Washington Soares, 1321
Fortaleza, Ceará, Brazil
acoelho@unifor.br

Thiago Monteiro Barbosa
Serviço Federal de
Processamento de Dados
Avenida Pontes Vieria ,832
Fortaleza, Ceará, Brazil
thiago.monteiro@serpro.gov.br

Abstract

Some software systems must respond to thousands or millions of concurrent requests. These systems must be properly tested to ensure that they can function correctly under the expected load. A common use of stress search-based testing is to find test scenarios that produce execution times that violate the timing constraints specified. The purpose of this paper is determine if hybrid algorithms are superior to single metaheuristics when solving stress testing problems and improve the process of stress testing with a tool that evolves the test model during its execution. The research proposes a hybrid metaheuristic approach that uses genetic algorithms, simulated annealing, and Tabu search algorithms for use in stress test models. A tool named IAdapter, a JMeter plugin used for performing search-based load, performance, or stress tests, was developed. Two experiments were conducted to validate the proposed approach. The first experiment was performed on an emulated component, and the second one was performed using an installed Moodle application. In both experiments, the use of a hybrid metaheuristic approach produced better fitness values.

1. Introduction

Many systems must support concurrent access by hundreds or thousands of users. Failure to scale users results in catastrophic failures and unfavorable media coverage [16].

The explosive growth of the Internet has contributed to the increased need for applications that perform at an appropriate speed. Performance problems are often detected late in the application life cycle, and the later they are discovered, the greater the cost to fix them [17].

The use of stress testing is an increasingly common practice owing to the increasing number of users. In this scenario, the inadequate treatment of a workload generated by concurrent or simultaneous access, generated by system users, can result in highly critical failures and negatively affect the customers's perception of the company [10] [16].

Stress testing determines the responsiveness, throughput, reliability, or scalability of a system under a given workload. The quality of the results of a system's load tests is closely linked to the implementation of the workload strategy. The performance of many applications depends on the load applied under different conditions. In some cases, performance degradation and failures arise only in stress conditions [13] [16].

A stress test uses a set of workloads that consist of many types of usage scenarios and a combination of different numbers of users. A load is typically based on an operational profile. Different parts of an application should be tested on various parameters and stress conditions [4]. The correct application of a stress test should cover most parts of an application above the expected load conditions (stress test)[10].

A stress test usually lasts for several hours or even a few days and only tests a limited number of workloads. The major challenge is to find the workloads that expose a major number of errors and to discover the maximum number of users supported by an application under test [5].

Search-based test is seen as a promising approach for verifying timing constraints [1]. A common objective of a load search-based test is to find test scenarios that produce execution times that violate the specified timing constraints [23].

We wish to determine if hybrid algorithms are superior to single metaheuristics when solving stress testing problem and and improve the process of stress testing with a tool that evolves the test model during its execution. The paper proposes the use of a hybrid metaheuristic approach that combines genetic algorithms, simulated annealing, and Tabu search algorithms in stress tests. A tool named IAdapter (www.iadapter.org), a JMeter plugin for performing search-based load tests, was developed. Two experiments were conducted to validate the proposed approach. The first experiment was performed on an emulated component, and the second one was performed using an installed Moodle application.

The remainder of the paper is organized as follows. Section 2 presents the research-proposed approach. Section 3 presents the IAdapter tool. Section 4 discusses the related work. Section 5 shows the results of two experiments performed using the IAdapter plugin. Conclusions and further work are presented in Section 6.

2. Improving Load, Performance, and Stress Search-Based Testing Using a Hybrid Metaheuristic Approach

A large number of researchers have recognized the advantages and huge potential of building hybrid mathematical programming methods and metaheuristics. The main motivation for creating hybrid metaheuristics is to exploit the complementary character of different optimization strategies. In fact, choosing an adequate combination of algorithms can be the key to achieving top performance in solving many hard optimization problems [21].

The proposed solution makes it possible to create a model that evolves during the test. The proposed solution model uses genetic algorithms, Tabu search, and simulated annealing in two different approaches. The three algorithms were selected because they can use a common genotype representation. The first approach uses the three algorithms independently, and the second approach uses the three algorithms collaboratively (hybrid metaheuristic approach).

In the first approach , the algorithms do not share their best individuals among themselves. Each algorithm evolves in a separate way (Fig. 1). The second approach uses the algorithms in a collaborative mode (hybrid metaheuristic). In this approach, the three algorithms share their best individuals found (Fig. 2).

The next subsections present details about the used metaheuristic algorithms (genotype representation and fitness function).

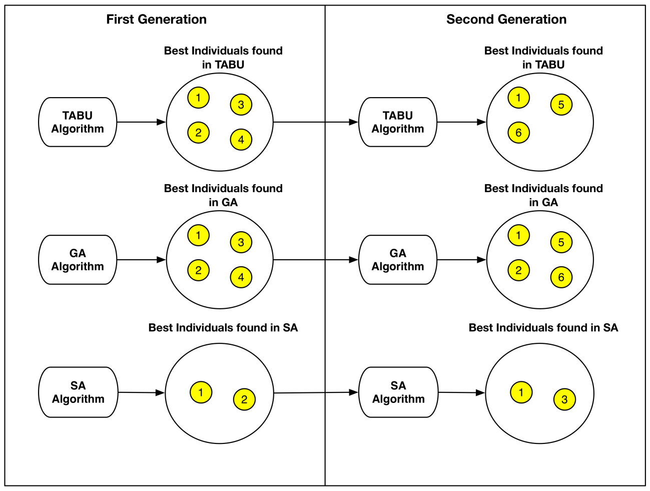
First Generation
Second Generation
Best Individuals found in TABU
TABU Algorithm
1
2
3
4
5
6
Best Individuals found in GA
GA Algorithm
Best Individuals found in SA
SA Algorithm

Figure 1. Use of the algorithms independently

2.1. Genotype representation

The genotype representation is composed by a linear vector with 23 genes. The first gene represents the name of an individual. The second gene represents the algorithm (genetic algorithm, simulated annealing, or Tabu search) used by the individual. The third gene represents the type of test. The next genes represent 10 scenarios and their numbers of users. Each scenario is an atomic operation: the scenario must log into the application, run the task goal, and undo any changes performed, returning the application to its original state.

Fig. 3 presents the genome representation and an example using the crossover operation. In the example, genotype

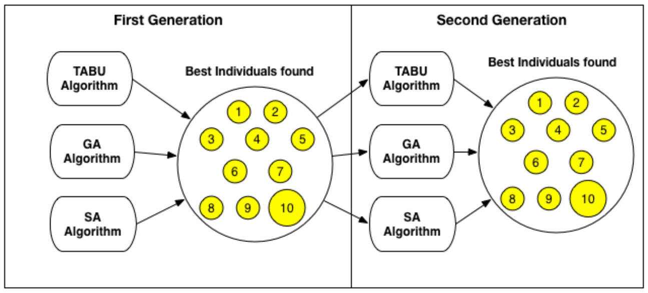

Figure 2. Use of the algorithms collaboratively

1 has the Login scenario with 2 users, the Form scenario with 0 users, and the Search scenario with 3 users. Genotype 2 has the Delete scenario with 10 users, the Search scenario with 0 users, and the Include scenario with 5 users. After the crossover operation, we obtain a genotype with the Login scenario with 2 users, the Search scenario with 0 users, and the Include scenario with 5 users.

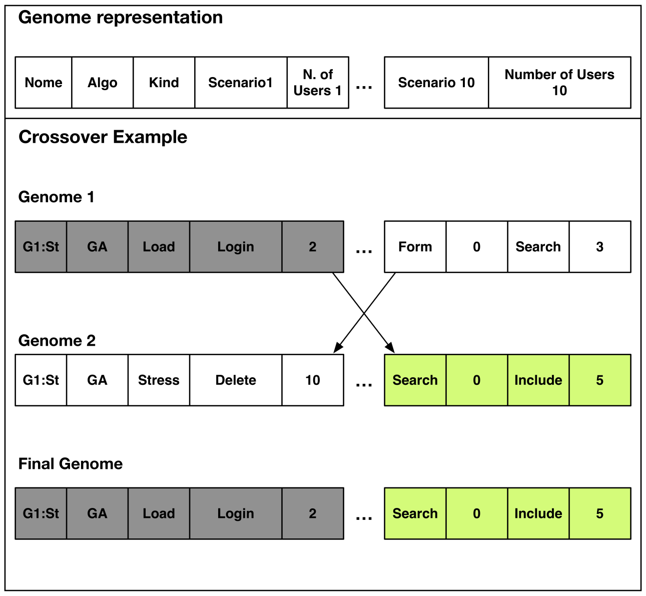

Figure 3. Genotype representation and crossover example

Fig. 4 shows the strategy used by the IAdapter tool to obtain the genotype of the neighbors for the Tabu search and simulated annealing algorithms. The neighbors are obtained by the modification of a single chromosome (scenario or number of users) in the genotype.

2.2. Objective (fitness) function

The proposed solution uses the following equation as fitness function:

$$\begin{aligned} fit = {} & 90percentileweigth * 90percentiletime \\ & +80percentileweigth * 80percentiletime \\ & +70percentileweigth * 70percentiletime + \\ & maxResponseWeigth * maxResponseTime + \\ & numberOfUsersWeigth * numberOfUsers - penalty \end{aligned} \quad (1)$$

The proposed solution uses 90,80 and 70-percentile. Some researchers advocate that the 90-percentile response

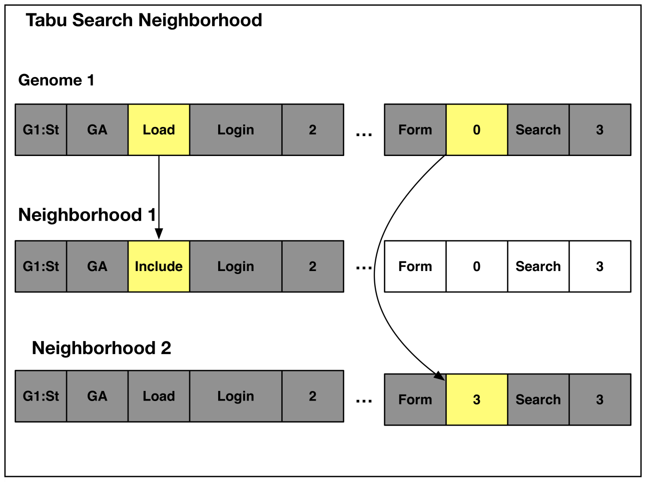

Figure 4. Tabu search and simulated annealing neighbor strategy

time is a better measurement than the average/medium response time, as the former accounts for most of the peaks, while eliminating the outliers [16]. The proposed solution's fitness function uses a series of manually adjustable user-defined weights (90percentileweight, 80percentileweight, 70percentileweight, maxResponseWeight, and numberOfUsersWeight). These weights make it possible to customize the search plugin's functionality. A penalty is applied when an application under test takes a longer time to respond than the level of service.

3. IAdapter

IAdapter is a JMeter plugin designed to perform search-based stress tests. The plugin is available on www.iadapter.org. JMeter is a desktop application designed to test and measure the performance and functional behavior of applications. The IAdapter plugin implements the solution proposed in Section 2.

JMeter has components organized in a hierarchical manner. The IAdapter plugin provides three main components: WorkLoadThreadGroup, WorkLoadSaver, and WorkLoadController.

WorkLoadThreadGroup is a component that creates an initial population and configures the algorithms used in IAdapter. The WorkLoadSaver component is responsible for saving all data in the database. The operation of the component only requires its inclusion in the test script. WorkLoadController represents a scenario of the test. All actions necessary to test an application should be included in this component. All instances of the component need to login to the application under test and return the application to its original state.

4. Related Work

The search for the longest execution time is regarded as a discontinuous, nonlinear, optimization problem, with the input domain of the system under test as a search space [23]. A common objective of search-based testing in stress tests is to find test scenarios that produce execution times that exceed the timing constraints specified. If a temporal error is found, the test was successful [23]. The application of evolutionary algorithms to stress tests involves finding the best- and worst-case execution times to determine whether timing constraints are fulfilled [1]. There are two measurement units normally associated with the fitness function in stress test: processor cycles and execution time. The processor cycle approach describes a fitness function in terms of processor cycles. The execution time approach involves executing the application under test and measuring the execution time [1] [24].

Table 1 shows a comparison between the presented research work and the research studies on load, performance, and stress tests presented by Afzal et al. [1]. Afzal's work adds to some of the latest research in this area ([11] [13] [8] [9] [3]). The columns represent the type of tool used (prototype or functional tool), and the rows represent the metaheuristic approach used by each research study (genetic algorithm, Tabu search, simulated annealing, or a customized algorithm). The table also divides the research studies by the type of fitness function used (execution time or processor cycles). Most research studies are limited to making prototypes of genetic algorithms. The presented research work is distinguished from others by having a functional tool using a hybrid approach.

The presented research work and Alesio's approach [3] use a hybrid approach with a functional tool. Whereas the present research uses an approach based on usage scenarios performing tests on an application installed in an available environment, Alesio uses sequence diagrams to select for arrival time of tasks in systems from safety-critical domains.

5. Experiments

This section presents two experiments. The first one was performed on an emulated component, and the second one was performed using an installed Moodle application. The experiments used the following fitness function:

Table 1. Distribution of the research studies over the range of applied metaheuristics

	Prototypes		**Functional Tool**
	Execution Time	Processor Cycles	Execution Time
GA + SA + Tabu Search			**IADAPTER**
GA	Alander et al.,1998 [2] Wegener et al., 1996 an 1997 [27][15] Sullivan et al., 1998 [23] Briand et al., 2005 [6] Canfora et al., 2005 [7]	Wegener and Grochtmann 1998 [26] Mueller et al., 1998 [18] Puschner et al. [22] Wegener et al., 2000 [28] Gro et al., 2000 [14]	Di Penta, 2007 [19] Garoussi, 2006 [11] Garousi, 2008 [12] Garousi, 2010 [13]
Simulated Annealing (SA)			Tracey, 1998 [25]
Constraint Programming			Alesio, 2014 [9] Alesio, 2013 [8]
GA + Constraint Programming			Alesio, 2015 [3]
Customized Algorithm		Pohlheim, 1999 [20]	

$$\begin{aligned} fit = 0.9 * 90percentiletime \\ +0.1 * 80percentiletime \\ +0.1 * 70percentiletime+ \\ 0.1 * maxResponseTime+ \\ 0.2 * numberOfUsers - penalty \end{aligned} \tag{2}$$

This fitness function intended to find individuals with the highest percentile of 90%, followed by individuals with a higher percentile time of 80% and 70%, maximum response time, and number of users. The first experiment implemented 27 generations, and the second experiment performed 6 generations, with 300 executions by generation (100 times for each algorithm), generating 300 new individuals. The experiments used an initial population of 100 individuals. The genetic algorithm used the top 10 individuals from each generation in the crossover operation. The Tabu list was configured with the size of 10 individuals and expired every 2 generations. The mutation operation was applied to 10% of the population on each generation.

5.1. First Experiment: Emulated Class Test

The first experiment aimed to perform performance, load, and stress testing on a simulated component. The purpose of using a simulated component was to be able to perform a greater number of generations in a shorter time available and eliminate variables such as the use of databases and application servers. The first experiment used a test class named SimulateConcurrentAccess. This class has a static variable named *x* and a set of methods that use the variable in a synchronized context.

Figure 5 presents the results obtained by the hybrid metaheuristic (HM) approach, genetic algorithm (GA), simulated annealing (SA), and Tabu search (TS) from 27 generations in the first experiment. The values are the maximum value of the fitness value obtained in each algorithm. A higher value in the table means that the scenario has a greater response time by the application under test. The signed-rank Wilcoxon non-parametrical procedure was used for comparing the results. The procedure showed that there was a significant improvement in the results using the three algorithms together.

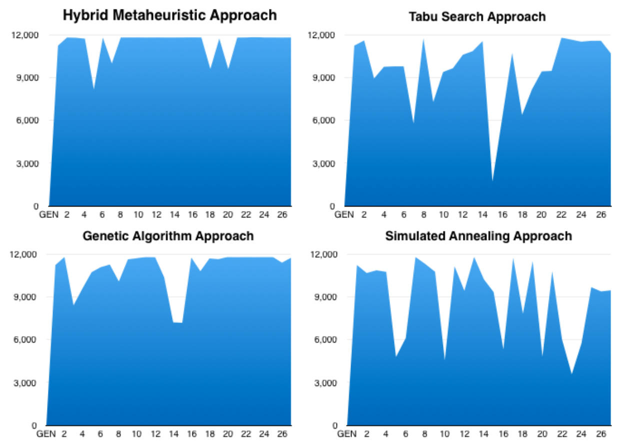

Figure 5. Maximum value of the fitness function by algorithm

5.2. Second Experiment: Moodle Application Test

The second experiment used a Moodle application installed in a machine with 500 GB of hard disk space and 8 GB of memory. The study used six application scenarios:

- PostDeleteMessage: This scenario posts and deletes messages in the Moodle application.
- MyHome: This scenario accesses the homepage of the user's application.
- Login: This scenario is responsible for user authentication by the application.
- Notifications: This scenario involves entering the notification page of each user.
- Start Page: This scenario shows the initial start page of the application.
- Badge: This scenario involves entering the badge page.

The maximum tolerated response time in the test was 30 seconds. Any individuals who obtained a time longer than the stipulated maximum time suffered penalties. The whole process of stress and performance tests, which took 3 days and about 1800 executions, was carried out without the need for monitoring by a test designer. The tool automatically selected the next scenarios to be run up to the limit of six generations previously established. The small number of samples of the experiment is insufficient to give a statistical significance to the results of the Wilcoxon procedure. However, it is noted that, in four of six generations, the collaborative approach presented the best values. The experiment succeeded in finding 29 individuals whose maximum time expected by the application was obtained. Table 2 shows an example of the six individuals with the highest fit values in the second experiment. The table shows the fitness value (Fit); the name of the scenario (Scenario); the number of users (Users); and the percentiles of 90%, 80%, and 70% (90per, 80per and 70per) in seconds.

Table 2. Example of individuals obtained in the second experiment

Id	Fit	Scenario	Users	90per	80per	70per
1	35800	MyHome	31	30	29	10
		Badges	4			
2	35795	MyHome	30	30	29	10
		Notifications	2			
		Badges	2			
3	35782	MyHome	32	30	29	10
		Badges	3			
4	35773	MyHome	22	30	29	10
		Notifications	6			
		Badges	9			
5	35771	MyHome	28	30	29	9
		Badges	6			
6	35683	MyHome	27	30	29	8
		Badges	10			

6. Conclusion

This paper presented a hybrid metaheuristic approach for use in load, performance, and stress testing. Two experiments were performed to validate the solution. The first experiment was performed on an emulated component, and the second experiment was performed using an installed Moodle application. The collaborative approach obtained better fit values in both experiments.

The main contributions of this research are as follows: The presentation of a hybrid metaheuristic approach for use in stress tests; the development of a JMeter plugin for search-based tests; and the automation of the load, performance, or stress test execution process. Among the future works of the research, the use of new combinatorial optimization algorithms such as very large-scale neighborhood search is one that we can highlight.

References

[1] W. Afzal, R. Torkar, and R. Feldt. A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, 51(6):957–976, 2009.

[2] J. T. J. Alander, T. Mantere, and P. Turunen. Genetic Algorithm Based Software Testing. In *Neural Nets and Genetic Algorithms*, 1998.

[3] S. D. I. Alesio, L. C. Briand, S. Nejati, and A. Gotlieb. Combining Genetic Algorithms and Constraint Programming. *ACM Transactions on Software Engineering and Methodology*, 25(1), 2015.

[4] C. Babbar, N. Bajpai, and D. Sarmah. Web Application Performance Analysis based on Component Load Testing. *International Journal of Technology*, 2011.

[5] C. Barna, M. Litoiu, and H. Ghanbari. Autonomic load-testing framework. *International conference on Autonomi*, pages 91–100, 2011.

[6] L. C. Briand, Y. Labiche, and M. Shousha. Stress testing real-time systems with genetic algorithms. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, page 1021, 2005.

[7] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. 2005., Canfora, G., An approach for QoS-aware service composition based on genetic algorithms.

[8] S. Di Alesio, S. Nejati, L. Briand, and A. Gotlieb. Stress testing of task deadlines: A constraint programming approach. *IEEE Xplore*, pages 158–167, 2013.

[9] S. Di Alesio, S. Nejati, L. Briand, and A. Gotlieb. Worst-Case Scheduling of Software Tasks – A Constraint Optimization Model to Support Performance Testing. *Principles and Practice of Constraint Programming*, pages 813–830, 2014.

[10] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth, and G. Weber. Realistic load testing of Web applications. In *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[11] V. Garousi. Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models using Genetic Algorithms. (August), 2006.

[12] V. Garousi. Empirical analysis of a genetic algorithm-based stress test technique. *Proceedings of the 10th annual conference on Genetic and evolutionary computation - GECCO '08*, page 1743, 2008.

[13] V. Garousi. A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation. *IEEE Transactions on Software Engineering*, 36(6):778–797, Nov. 2010.

[14] H. Gross, B. F. Jones, and D. E. Eyres. Structural performance measure of evolutionary testing applied to worst-case timing of real-time systems. *Software, IEE Proceedings-*, 147(2):25–30, 2000.

[15] B. J. J. Wegener, K. Grimm, M. Grochtmann, H. Sthamer. Systematic testing of real-time systems. *EuroSTAR'96: Proceedings of the Fourth International Conference on Software Testing Analysis and Review*, 1996.

[16] Z. Jiang. *Automated analysis of load testing results*. PhD thesis, 2010.

[17] I. Molyneaux. *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. "O'Reilly Media, Inc.", 1st edition, Jan. 2009.

[18] F. Mueller and J. Wegener. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No.98TB100245)*, 1998.

[19] M. D. Penta, G. Canfora, and G. Esposito. Search-based testing of service level agreements. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097, 2007.

[20] H. Pohlheim, M. Conrad, and A. Griep. Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences. *Analysis*, (724):804—-814, 2005.

[21] J. Puchinger and R. Raidl. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization : A Survey and Classification. *Artificial Intelligence and Knowledge Engineering Applications a Bioinspired Approach*, 3562:41–53, 2005.

[22] P. Puschner and R. Nossal. Testing the results of static worst-case execution-time analysis. *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, 1998.

[23] M. O. Sullivan, S. Vössner, J. Wegener, and D.-b. Ag. Testing Temporal Correctness of Real-Time Systems — A New Approach Using Genetic Algorithms and Cluster Analysis —. pages 1–20.

[24] N. J. Tracey. *A search-based automated test-data generation framework for safety-critical software*. PhD thesis, Citeseer, 2000.

[25] N. J. Tracey, J. a. Clark, and K. C. Mander. Automated Programme Flaw Finding using Simulated Annealing. 1998.

[26] J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.

[27] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, 1997.

[28] H. Wegener, Joachim and Pitschinetz, Roman and Sthamer. Automated Testing of Real-Time Tasks. *Proceedings of the 1st International Workshop on Automated Program Analysis, Testing and Verification (WAPATV'00)*, 2000.