Computers **2013**, *2*, 67–87; doi:10.3390/computers2020067

OPEN ACCESS

computers

ISSN 2073-431X

www.mdpi.com/journal/computers

Article

A Characterization of the Utility of Using Artificial Intelligence to Test Two Artificial Intelligence Systems

Jeremy Straub * and Justin Huber

Department of Computer Science, University of North Dakota, 3950 Campus Road, Stop 9015, Grand Forks, ND 58202-9015, USA; E-Mail: justin.r.huber@my.und.edu

***** Author to whom correspondence should be addressed; E-Mail: jeremy.straub@my.und.edu; Tel.: +1-701-777-4107; Fax: +1-701-777-3330.

Received: 28 December 2012; in revised form: 4 February 2013 / Accepted: 22 May 2013 / Published: 31 May 2013

Abstract: An artificial intelligence system, designed for operations in a real-world environment faces a nearly infinite set of possible performance scenarios. Designers and developers, thus, face the challenge of validating proper performance across both foreseen and unforeseen conditions, particularly when the artificial intelligence is controlling a robot that will be operating in close proximity, or may represent a danger, to humans. While the manual creation of test cases allows limited testing (perhaps ensuring that a set of foreseeable conditions trigger an appropriate response), this may be insufficient to fully characterize and validate safe system performance. An approach to validating the performance of an artificial intelligence system using a simple artificial intelligence test case producer (AITCP) is presented. The AITCP allows the creation and simulation of prospective operating scenarios at a rate far exceeding that possible by human testers. Four scenarios for testing an autonomous navigation control system are presented: single actor in two-dimensional space, multiple actors in two-dimensional space, single actor in three-dimensional space, and multiple actors in three-dimensional space. The utility of using the AITCP is compared to that of human testers in each of these scenarios.

Keywords: artificial intelligence; testing artificial intelligence systems; using artificial intelligence for testing; artificial intelligence test case generation

---

1. Introduction

Validation of the safe performance of an artificial intelligence system (AIS), which operates in close proximity to humans or which could prospectively injure humans through its failure or maloperation is an integral part of the system testing process. Failure to perform proper validation can result in system failure or catastrophe, as described in [1]. While conventional testing techniques can validate that software components operate as designed on a unit and assembly basis, this does not establish that (1) the system will work as desired in a real-world environment and (2) that the system will perform safely or effectively under the multitude of prospective operating conditions that it may encounter. These can be effectively validated via the creation and implementation of numerous test cases. This, however, is a tedious and time-consuming task, if performed manually. The time required for manual testing, combined with resource limitations, may result in insufficient testing being performed and critical defects going unidentified.

The use of a simple artificial intelligence test case producer (AITCP) to validate an AIS is proposed and evaluated in this paper. The AITCP is able to simulate numerous maneuvers of human or human controlled actors at a rate significantly exceeding that of an actual human. This allows significantly more scenarios to be generated and executed, within a given testing period. As the automated testing can be run, in many cases, without human supervision, the amount of time available for testing may be significantly increased (e.g., overnight when human testers are not at work).

The AITCP could start from a human generated test scenario and makes changes to it based upon a modification algorithm (e.g., ant colony optimization [2], genetic approaches [3], *etc.*). For the testing described herein, a collision avoidance AIS is tested via the creation of testing maneuvers (generated in response to AIS maneuvers). The AITCP attempts to position its actor or actors to cause a collision into it by the AIS-operated craft (it attempts to avoid a self-caused collision). The AITCP utilized herein is a very simple form of artificial intelligence; it (similar to many swarm intelligence approaches) creates a complicated pattern of behavior from a very simple algorithm. In this case, the AITCP attempts to place itself in as close proximity to the AIS actor as possible, without causing a collision itself.

2. Background

To ensure the success of an autonomous system, an efficient method of validation is required. A sufficient number of test cases need to be developed in order to show that the system can perform acceptably in the real world environment in which it was designed to operate. There is a significant body of research related to validation and test case generation techniques for artificial intelligence systems and their evaluation. Existing work in four areas (testing artificial intelligence systems using test cases, artificial intelligence-based test case generation, testing as a search problem, and software and artificial intelligence failures) is now reviewed.

2.1. Testing Artificial Intelligence (AI) with Test Cases

One of the most basic forms of testing an autonomous system is with manually generated test cases. This involves a human tester creating scenarios that will be presented to the AI, or under which the performance of the AI will be evaluated. Felgenbaum [4], for example, reviewed artificial intelligence

systems designed for diagnosis based on medical case studies and concluded that the modularity of the "Situation → Action" technique allowed for rules to be changed or added easily as the expert's knowledge of the domain grew. This allowed more advanced cases to be used for validation.

Chandrasekaran [5] suggests that the evaluation of an AI must not be based only on the final result. In [5], an approach to the validation of Artificial Intelligence Medical (AIM) systems for medical decision-making is presented. The paper also examines some of the problems encountered during AIM evaluations. During performance analysis of AI systems, evaluating success or failure based upon the final result may not show the entire picture. Intermediate execution could show acceptable results even though the final result is unsatisfactory. Evaluating important steps in reasoning can help alleviate this issue.

Another example of AI testing with test cases is presented by Cholewinski *et al.* [6] who discuss the Default Reasoning System (DeReS) and its validation through test cases derived from TheoryBase, a benchmarking system "designed to support experimental investigations of nonmonotonic reasoning systems based on the language of default logic or logic programming". Through the use of TheoryBase-generated default theories, DeReS was shown to be a success. Cholewinski *et al.* also proffer that TheoryBase can be used as a standalone system and that any non-monotonic reasoning system can use it as a benchmarking tool.

Brooks [7] comments on the use of simulation testing. In [7] the possibility of controlling mobile robots with programs that evolve using artificial life techniques is explored. Brooks has not implemented or tested the ideas presented; however, some intriguing notions regarding simulation and testing of physical robots are discussed. Using simulated robots for testing, before running the programs on physical robots, has generally been avoided for two reasons [8–10]: First, for real-world autonomous systems, there is a risk that the time involved in resolving issues identified in a simulated environment will be wasted due to dissimilarities between the types of events occurring in simulation *versus* the real operating space. Second, emulating real-world dynamics in a simulated environment is difficult due to differences in real world sensing. This increases the chance of the program behaving differently in the real world. The use of simulated robots for testing may uncover basic issues impairing a control program. However, this approach tends not to uncover some problems encountered when tested in a real-world environment.

The previous studies have related to validation methods and test cases used to assess AI systems designed for practical or complex tasks in everyday life. Billings *et al.* [11], alternately, explores the use of an AI designed for competition rather than the performance of particular jobs. Poki is an AI driven program built as an autonomous substitute for human players in world-class poker tournaments, (specifically, Texas Hold'em tournaments). In Poker, players are constantly adapting across playing many hands. Two methods are discussed to validate the program: self-play and live-play.

Self-play tests are a simple method of validation where an older version of the tested program is pitted against the current version. This allows a great variety of hands to be played in a short amount of time. Live-play tests seek to alleviate this problem and are, thus, considered by Billings *et al.* as essential for accurate evaluation. Implementing the poker AI as part of an online game is one of the more effective ways to test performance, as thousands of players are able to play at any given time.

In testing Poki, Billings *et al.* tested each version of the program for 20,000 hands using the average number of small bets won per hand as a performance measurement before translating the results.

The work done on the Poki poker system shows that validating an AI through testing its function against another AI (itself in this case) is a helpful tool for evaluating system performance. Real world application test cases are also shown to be critical in validating the utility of the AI-based validation process.

2.2. AI Test Case Generation

While manual test case generation may be suitable for a system where the scope of performance is limited, systems that have to operate in a real-world environment must function under a large variety of systems. Given this, a more efficient approach to test case generation is desirable.

Dai, Mausam and Weld [12] deal with a similar problem, except in the context of evaluating human performance on a large scale. They look at using an AI adaptive workflow, based on their TurKontrol software, to increase the performance of a decision making application. Their workflow controller is trained with real world cases from Amazon's Mechanical Turk. Mechanical Turk utilizes humans to perform repetitive tasks such as image description tagging. For this, a model of performance and iterative assessment is utilized to ensure appropriate quality. Through autonomously determining whether additional human review and revision was required, TurKcontrol was able to increase quality performance by 11%. Dai, Mausam, and Weld note that the cost of this increased performance is not linear and that an additional 28.7% increase in cost would be required to achieve a level of comparable performance.

The work performed by Dai, Mausam, and Weld provides an implementation framework for autonomously revising AI performance, based upon their work in assessing and refining human performance. The approach can be extended to incorporate AI workers and evaluators, for applications where these tasks can be suitably performed autonomously.

Pitchforth and Mengersen [13] deal with the problem of testing an AI system. Specifically, they look at the process of validating a Bayesian network, which is based on data from a subject matter expert. They note that previous approaches to validation either involved the comparison of the output of the created network to pre-existing data or relied upon an expert to review and provide feedback on the proposed network. Pitchforth and Mengersen proffer, however, that these approaches fail to fully test the networks' validity.

While Pitchforth and Mengersen do not provide a specific method for the development of use and test cases, their analysis of the validation process required for a Bayesian network informs the process of creating them. It appears that use cases are relevant throughout their validation framework and test cases are specifically relevant to the analysis of concurrent and predictive validity. Moreover, the convergent and divergent analysis processes may inform the types of data that are required and well suited for test case production.

The use of AI in software development and debugging is also considered by Wotawa, Nica, and Nica [14], who discuss the process of debugging via localizing faults. Their proposed approach, based on model-based diagnosis, is designed to repetitively test a program or area of code to determine whether

it functions properly. To this end, they propose an approach that involves creating base test cases and applying a mutation algorithm to adapt them.

While Wotawa, Nica, and Nica's work is quite limited (as they note) in the context of line-by-line review of program code, the fundamental concept is exceedingly powerful. Input parameters can be mutated extensively without having to create a mechanism to generate an associated success condition.

AdiSrikanth *et al.* [15], on the other hand, deal with a more generalizable approach. They propose a method for test case creation based upon an artificial bee colony algorithm. This algorithm is a swarm intelligence approach where three classes of virtual bees are utilized to find an optimal solution: employed, onlookers, and scouts. Bees seek to identify "food sources" with the maximum amounts of nectar.

In the implementation for optimizing test cases, a piece of code is provided to AdiSrikanth *et al.*'s tool. This software creates a control flow graph, based on the input. The software then identifies all independent paths and creates test cases, which cause the traversal of these paths. Optimization is achieved via a fitness value metric.

This work demonstrates the utility of swarm intelligence techniques for test case generation and refinement. AdiSrikanth *et al.*, regrettably, fail to consider the time-cost of their proposed approach. While an optimal solution for a small program can be generated fairly quickly, the iterative approach that they utilize may be overly burdensome for a larger codebase.

Similar to the bee colony work performed by AdiSrikanth *et al.* is the ant colony optimization work performed by Suri and Singhal [2]. Suri and Singhal look at using ant colony optimization (ACO) for performing regression analysis. Specifically, they look at how regression tests should be prioritized to maximize the value of regression testing, given a specific amount of time to perform the testing within.

The time requirements for ACO-selection-based execution ranged between 50% and 90% of the time required to run the full test suite. It appears that the average is around the 80% mark.

A more general view is presented by Harman [16], who reviews how artificial intelligence techniques have been used in software engineering. He proffers that three categories of techniques have received significant use: optimization and search, fuzzy reasoning, and learning. The first, optimization and search, is utilized by the field of "Search Based Software Engineering" which converts software engineering challenges into optimization tasks. Fuzzy reasoning is used by software engineers to consider real-world problems of a probabilistic nature. Finally, with "Search Based Software Engineering" (SBSE) the wealth of solution-search knowledge in the AI optimization domain is brought to bear on software engineering problems. Harman proffers that the continued integration of AI techniques into software engineering is all but inevitable, given the growing complexity modern programs.

2.3. Testing as a Search Problem

Validation can also be conceived of as a search problem. In this case, the search's 'solution' is a problem in the system being tested. Several search approaches relevant to this are now reviewed. The use of these approaches in testing in most cases remains to be tested and future work in this area may include the comparison of these approaches and their evaluation in terms of performance across various testing applications.

Pop *et al.* [17], for example, present an enhancement of the Firefly search algorithm that is designed to elicit optimal, or near-optimal, solutions to a semantic web service composition problem. Their approach combines the signaling mechanism utilized by fireflies in nature with a random modification approach.

Pop *et al.* compare the firefly solution with a bee-style solution. The bee-style solution took 44% longer to run, processing 33% more prospective solutions during this time. The firefly approach had a higher standard deviation (0.007 *versus* 0.002). Pop *et al.* assert that their work has demonstrated the feasibility of this type of approach.

Shah-Hosseini [18] presents an alternate approach, called the Intelligent Water Drop (IWD) approach, to problem solving that utilizes an artificial water drop with properties mirroring water drops in nature. Two properties of water drops are important. The first important aspect is its soil carrying capability. The water drops, collectively, pick up soil from fast-moving parts of the river and deposit it in the slower parts. Second, the water drops choose the most efficient (easiest) path from their origin to their destination. The IWD method can be utilized to find the best (or near-best) path from source to destination. It can also be utilized to find an optimal solution (destination) to a problem that can be assessed by a single metric. Duan, Liu, and Wu [19] demonstrate the IWD's real-world application in the application of route generation and smoothing for an unmanned combat aerial vehicle (UCAV).

Yet another search technique is presented by Gendreau, Hertz, and Laporte [20] who discuss an application of a metaheuristic improvement method entitled the Tabu Search, which was developed by Glover [21,22]. This approach takes its name from the use of a 'Tabu List', which prevents redundant visits to recently visited nodes via placing them on a list of nodes to avoid. The approach is open-ended and allows exploration of less-optimal-than-current solutions to allow the search to leave local minimums in search of the global minimum.

Fundamentally, as an improvement method, the Tabu Search visits adjacent solutions to the current solution and selects the best one to be the new current solution. Because of this, it can be initialized with any prospective solution (even an infeasible one). Gendreau, Hertz, and Laporte evaluate this search in the context of TABUROUTE, a solution to the vehicle routing problem. They conclude that the Tabu Search outperformed the best existing heuristic-based searches and that it frequently arrives at optimal or best known solutions.

Finally, Yang and Deb [23] propose a search based upon the egg-laying patterns of the cuckoo. This bird lays eggs in the nests of other birds of different species. The bird that built the nest that the cuckoo lays its egg in may, if it detects that the egg is not its own, destroy it or decide to abandon the nest. The Cuckoo Search parallels this. Each automated cuckoo creates an egg, which is a prospective problem solution, which is placed into a nest at random. Some nests that have the generation's best solutions will persist into the next generation; a set fraction of those containing the worst performing solutions will be destroyed. There is a defined probability of each nest being destroyed or the egg removed (paralleling the discovery of the egg by the host bird in nature). New nests are created at new locations (reached via Levy flights) to replace the nests destroyed (and maintain the fixed number of nests).

Walton, Hassan, Morgan, and Brown [24] refine this approach. Their Modified Cuckoo Search incorporates two changes designed to increase the speed of convergence at an optimal solution. First, they change the distance of the Levy flight from a fixed value to a value that declines on a generation-by-generation basis, with each generation having a value that is the initial value divided by the square

root of the generation ($V_I/\sqrt{G}$). Second, they create a mechanism to seed new eggs that are based upon the best currently known performing eggs. To do this, a collection of top eggs is selected and two of these eggs are selected for combination. Walton, Hassan, Morgan, and Brown assert that the Modified Cuckoo Search outperformed the Cuckoo Search in all test cases presented and that it also performed comparably to or outperformed the Particle Swarm Optimization approach.

Bulatovic, Dordevic, and Dordevic [25] demonstrate the utility of the Cuckoo Search to real-world problems. They utilize it to optimize 20 design variables as part of solving the six-bar double dwell linkage problem in mechanical engineering. Gandomi, Yang, and Alavi [26] demonstrate its utility on a second set of real-world problems related to design optimization in structural engineering.

2.4. Software and AI Failures

The need for verification and validation of AI systems is now reviewed. Clearly, not all systems require the same level of extensive validation. The impact and likelihood of systems' failure are key considerations in determining how much testing and other validation is required. Halawani [1] proffers that too much reliance is placed in software, including artificial intelligence control systems. Several examples of highly impactful failures illustrate and support this. A nearly-catastrophic error occurred in 1983 when software running a Soviet early warning system misidentified sunlight reflection from clouds as a prospective U.S. missile strike [1,27]. The Mars Climate Orbiter, a $125 million dollar spacecraft, crashed due to a units mismatch between two systems [1,28]. One was using and expecting metric units, while the other used and expected imperial units. A similar (easily correctable if caught) issue resulted in the loss of the Mariner I probe [1,29]. The initial 1996 launch of the Ariane 5 rocket failed due to an integer conversion issue at a cost of approximately one-half billion dollars [1,27]. A radiation therapy machine, the Therac-25, subjected two patients to lethal doses of radiation during a treatment [1,30].

3. The Need for AI Validation & Verification

The validation of artificial intelligence systems can be sub-divided into five categories: unit testing, integration/assembly testing, functional validation, performance validation, and the verification of safe operations. Unit testing utilizes conventional methods to validate the performance of individual system components and their combined operation. Integration testing can utilize standard approaches such as the top-down, bottom-up, and sandwich approaches; however, these approaches do little more than produce a unit-tested assembly (and track bugs back to their module of origin for repair). Functional validation assesses whether the AIS performs as desired in a real-world environment. Performance validation verifies that the speed and other characteristics of system performance meet requirements. This is particularly important for a system that must operate in real-time, as a decision that cannot be made fast enough may result in catastrophe. Finally, the verification of safe operations involves assessing whether the system avoids damage or injury to its operating environment and other actors (e.g., humans, human operated craft) in it. These last three categories of AIS validation and verification are aided by the AITCP and are discussed, in more detail, hereafter.

3.1. Functional Validation

Functional validation (also known as functional testing) is based on the concept that a program is effectively equivalent to a mathematical function that maps from each member of a set of inputs to one or more corresponding outputs [31]. Basic functional validation can be performed using human created test cases. For a prototypical system, these test cases generally are created to validate response to inputs representing each condition and boundary regions to ensure that system response is correct. The real-world operating environment, however, renders this simplistic testing approach impractical. The combination of the numerous conditions possible for each of the numerous system elements results in an impossibly large number of combinations that would be required to completely test the system. When ambiguous elements are added (e.g., sensors perceiving the real-world environment), the number of required test cases required approaches infinity. The AITCP, while not able to test every combination, maximizes the use of available testing time via making and rapidly testing modifications to a base human-generated strategy. This allows multiple permutations of numerous strategies to be effectively tested and re-tested for regression purposes.

3.2. Performance Validation

Performance validation (also known as performance testing) seeks to define, quantitatively or qualitatively, the desired performance of a system and compare actual system performance to this standard. For quantitative measures (e.g., a maximum response time to a particular type of input) successful attainment can determined reasonably easily. For performance measures that do not vary significantly, based on inputs (e.g., a function that requires a similar amount of time to run in all cases) compliance can be determined easily via manual testing. Automated testing, such as with the AITCP, can aid in determining compliance with performance standards, across a wide variety of scenarios, for metrics that vary significantly, based on inputs.

Validating compliance with some qualitative performance standards can be significantly more difficult, as it may require the comparison and juxtaposition of two measures. For example, a standard requiring timely (e.g., within the amount of time between perception and required response) response to input, will require a determination of both the amount of time allowed for the response and the amount of time required for making the decision. The AITCP can be utilized, within an appropriate simulation environment, to allow determination of these metrics and the resulting compliance status.

3.3. Verification of Safe Operations

The verification of safe operations is a special case of performance validation. It is based on the definition of a set of qualitative standards defining safe performance for the system and compliance testing to validate achievement of these standards. Unlike most performance metrics, however, the verification of safe performance is not based on an average. Ideally, it would be based on the generation and implementation of an exhaustive set of test cases. However, this is problematic in most instances, as the prospective test space is too large fully cover with automated (much less manual) testing. An operating presumption that any prospective combination of conditions could result in a safe operations violation dictates that the test space be explored as fully as possible, given resource

constraints. Ideally, this exploration would not be based on a random or otherwise uninformed approach. Human testers offer a solution to this as they allow focus to be directed to test space areas that exhibit the most likelihood of violation generation (e.g., by following paths that appear to be approaching a violation condition until a violation occurs or the metric begins moving away from the violation boundary). Genetic algorithms, and other AI techniques, allow the autonomous implementation of similar techniques. An AITCP incorporating this type of logic could allow a utility-maximizing exploration of the test space within time and other resource limitations, for metrics where this is appropriate. For metrics that do not exhibit this type of a pattern, the AITCP approach allows a random or predefined pattern-based exploration of the space to attempt to identify locations of condition breach.

4. Using AI to Test AI

An AIS, operating in a real-world environment, will encounter numerous scenarios that cannot be effectively preconceived by a human tester. An alternate approach to humans trying to predict scenarios that may cause the AIS to fail is to try a lot of different scenarios, quickly. This section reviews the value of using an AI tester for testing an AIS and presents an initial qualitative evaluation of the efficacy of an AI testing regime for an AIS.

4.1. Value of Using AI for AI System Testing

The use of an artificial intelligence testing mechanism for testing any system (including another artificial intelligence) could create four areas of prospective benefit. These include reducing the need for human involvement in testing, increasing in the amount of testing performed for a given level of time and other resources, increasing the objectivity of testing, and performing tests that are not conducive to human performance. Each prospective source of value will now be discussed.

Reducing the need for human involvement in certain aspects of testing can reduce testing costs or free resources for performing other types of testing that humans are particularly well suited for. For example, humans freed from regression testing (which was automated) might instead be tasked with creating scenarios that are designed to attempt to break particular aspects of the code. The intuition as to where certain types of errors occur, due to common programming practices and creation of combination-style scenarios that may test particular conditions of multiple units concurrently are an area where humans can add particular value (as AI-based testing for these is not presently possible, in many cases).

Increasing in the amount of testing performed for a given level of time and other resources is possible due to the comparatively faster speed of testing for AI *versus* human testers, in certain scenarios. For example, the work presented herein (which is turn-based) allows the two AIs (the system under test and AI tester) to move as fast as they are able to. Other testing scenarios (e.g., performing the same test with physical robots or in a simulation environment that could not or did not allow faster-than real-time operation) may not be as suitable for an increased testing rate. However, in these cases, it may be possible to perform testing in parallel (as human operators are not needed for each station, *etc.*) and still achieve significantly superior tests-performed-per-unit-time results.

Increasing the objectivity of testing is possible using automated AI testers, as many performance metrics can be influenced by the performance of the tester. For example, a tester that performed inferiorly at predicting what direction the AI was going, in the example used herein, might never discover a bug

that was present (as most of his or her time was spent trying to catch/keep up with the AI, instead of devising test scenarios for it). This objectivity is particularly important if the number of new bugs being detected as a function of time is tracked as part of the decision making process about when to end testing.

Performing tests that are not conducive to human performance is another key use for an AI tester. In the work presented herein, it was discovered that it was very difficult for humans to control the three-dimensional actors. This is likely due to issues with perceiving the exact (and relative) height of the AI and human actors. No solution, short of implementing the exact algorithm that the AI tester uses (always moving towards the other AI unless that movement will cause an at-fault collision), was found for this. Further, human implementation of this algorithm is simply wasteful (as the humans are not as efficient as the computer in doing so). Moreover, the implementation of this algorithm (which requires assessing each dimension separately, for most operators) removes the fluid feel of the testing (which may preclude intuitive-style testing scenarios being created by testers).

4.2. Qualitative Evaluation of Using AI to Test AI Systems

Qualitative evaluation of the benefits and drawbacks of using an AITCP to test an AIS are now presented (Section 7 presents and Section 8 analyzes quantitative results). This qualitative evaluation fundamentally involves comparing the positives and negatives of the AITCP approach and the approximate cost differential created by this approach.

The majority of the benefits of the AITCP testing approach were enumerated in the above Section 4.1 and their details will not be enumerated again here. The drawbacks of the approach are triggered by removing the human from the testing loop. This eliminates the potential for a human tester to intuitively, intentionally, subliminally or otherwise, devise scenarios to test prospective system issues that might not be arrived at autonomously by the AI. This potential for making a random connection can be utilized via specific testing activities related to these connections. However, the additional time and familiarity with the system created by repetitive testing cannot be simulated with limited sessions to try to devise innovate testing scenarios. This could result in a bug that might be discovered via intuitive human-driven testing going undetected by automated testing.

It is necessary to, thus, offset this unquantifiable potential against the tangible benefits of the magnitude of testing that can be conducted autonomously (given a set level of time) and the decreased cost from not having to incur the costs of human testers. No ‘intuitive connection’ testing scenarios identified bugs in the believed-good AIS and the level of bug reports was not increased for the AIS with known issues, for the current work. Moreover, the thought process associated with this introduced a small testing performance penalty. However, this behavior may not be typical for all applications and an ‘intuitive connection’ test may, in some cases, identify critical defects that might not be otherwise identified.

5. A Set of Initial Experiments in Using AI to Test AI

A set of experiments was performed to evaluate the utility of using an AITCP for testing an AIS. This section describes the experimental design and methods used.

5.1. Experimental Design

Eight experimental conditions were created based on two options for each of three experimental variables. These variables were the number of AI actors controlled by the AITCP, the operating environment (two-dimensional *versus* three-dimensional), and the state of the system under test (believed-good *vs.* known-bad). The number of actors controlled by the AITCP varied between one and two. The operating environment varied between a two-dimensional environment where a control AI for a surface-rover-style robot was tested and a three-dimensional environment where a control AI for an unmanned aerial vehicle (UAV) was tested. In addition to the additional movement flexibility, the three-dimensional scenario incorporates a requirement that the AI being tested move continuously.

The testing performance of a believed-good AI was compared to the testing performance of an AI with an intentionally introduced defect. The detection rate of faults and the testing duration were collected for each condition.

5.2. Experimental Methods

Multiple test runs of 2,000 tests comprised of 2,000 moves each were conducted for each experimental condition. The amount of time required for a human tester to conduct a similar number of tests was extrapolated, based on performing a subset of this number of test runs. The fault detection rate between human testing and testing performed by the AITCP was also conducted. From these two metrics, the utility of the two approaches was calculated and compared. Human testing was conducted (see [24]) for two-dimensional, one actor condition. Limited testing was conducted for the two-dimensional, two-actor scenario (to assess comparative performance, given that performance was shown to be relatively consistent from the initial two-dimensional one-actor scenario). Due to limitations of human perception of a three-dimensional environment on a two-dimensional screen, human control was ineffective (in achieving the goal of causing a not-at-fault collision) under the testing conditions. Prospective human performance was, thus, extrapolated from the two-dimensional conditions, for comparison purposes.

6. Implementation of AI Testing of AI System

A demonstration of the utility of utilizing an AI to test an AI system requires several components, which are discussed now. These include a testing environment, an AI system to test and an AI actor-simulator to simulate other actors (e.g., people in two-dimensional simulations, aircraft in three-dimensional simulations) within the test environment.

6.1. AI System Under Test

The AISUT comprises a basic maneuvering and avoidance system. The system is designed to operate a robot (two-dimensional) or UAV (three-dimensional) within a defined space (Figure 1, for example, shows the space that was utilized for the two-dimensional testing, a plus-shaped area) while avoiding any obstacles presented (either fixed or mobile). The movements (which in a real-world simulation would be task-driven) are simulated via the selection of a random initial direction of travel.

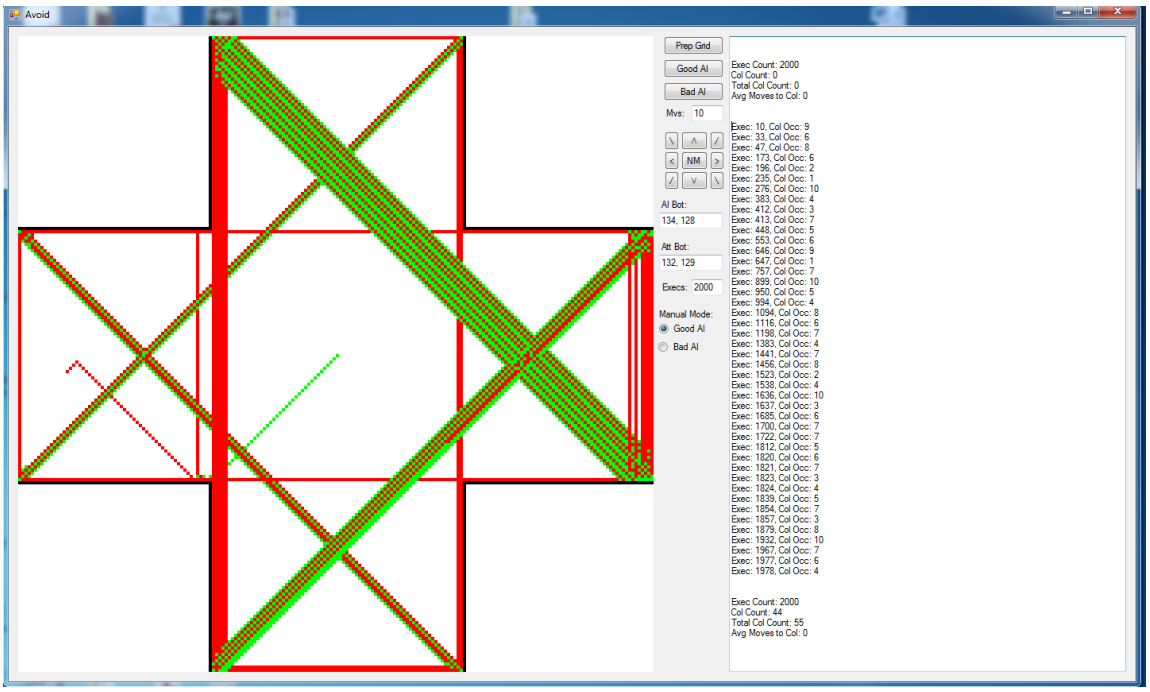
A new direction is selected if a wall or other impediment is detected; there is also a small probability of a random direction change during each movement turn.

6.2. Testing Environment

A testing environment was created to evaluate the relative performance of an AI actor-controller simulating the actions of a human operator *versus* an actual human operator. The testing environment for the two-dimensional, single actor simulation is shown in Figure 1. In this diagram, the red dots indicate the position of the testing actor and the green dots indicate the position of the actor or actors controlled by the AI system under test (AISUT). The AI actor-controller has complete knowledge of the operating environment and the position of the AISUT actor within it. The AISUT has no knowledge of the environment, beyond the squares directly adjacent to it. A model for simulating sensor failure (or non-interpretable data, *etc.*) was not created. This is seen as a topic for future work and could easily be incorporated into the simulation environment. Human-controlled actors are moved via the arrow buttons towards the top center of the screen shown in Figure 1.

The system utilized for the three-dimensional testing is shown in Figure 2. This system is very similar to the two-dimensional version shown in Figure 1. However, in this instance, the colors (shades of green and red) vary to indicate the height of the actor. The lowest positions are colored in dark red and green and the highest positions are covered in light red and green.

Figure 1. Artificial intelligence (AI) testing environment for two-dimensional exercises.

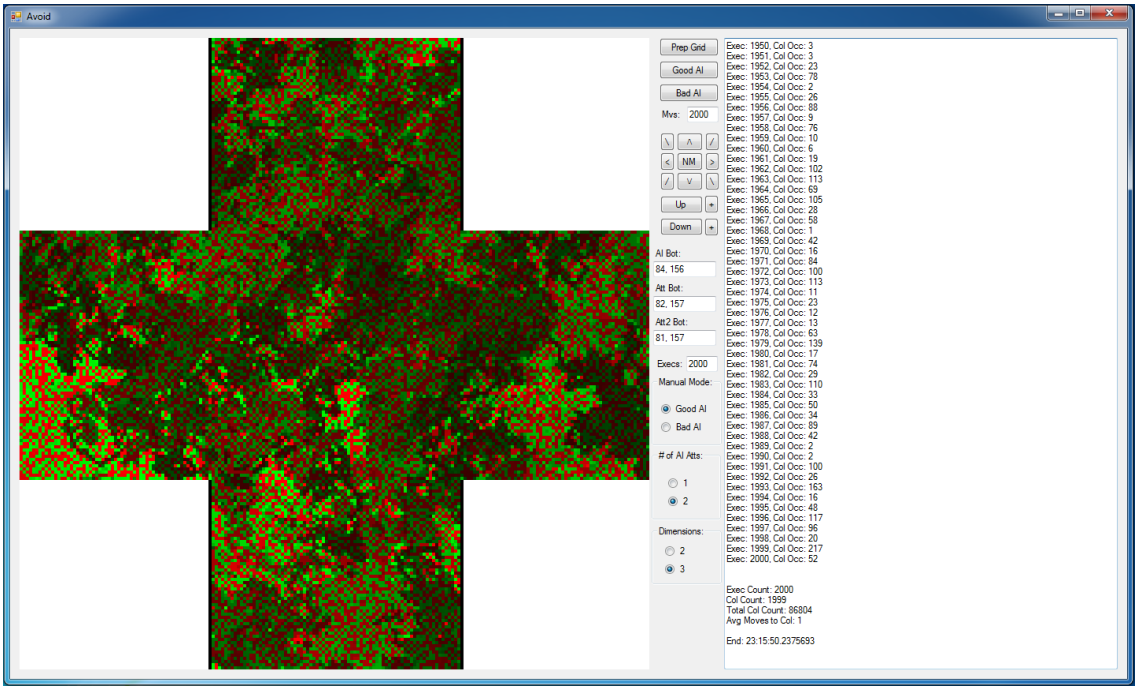


6.3. AI Actor Simulator

An AI test routine (AITR) was created for each of the four experimental scenarios which attempts to place its actors in a position (or in positions, for multiple-actor scenarios) to attempt to cause the AISUT to cause a collision. Note that the test routine will not attempt to cause its actors to collide with the

AISUT's position (as adversary avoidance is not the point of the testing, verifying safe operations is), which would be much easier. For both single actor and multi-actor scenarios, this involves maneuvering the actors to locations adjacent to that of the AISUT's position. In multi-actor scenarios, the multiple actors will, obviously, not be allowed to occupy the same adjacent-to-AISUT position; however, no special logic exists to attempt to position these units to particularly impair movement, *etc.* This is a potential topic for future work.

Figure 2. AI testing environment for three-dimensional exercises.



In each instance, the AISUT and AITR actors begin in separate locations. The AITR actors are maneuvered towards the position of the AISUT's position in the most efficient manner possible. Once in close proximity, they mirror the movement of the AISUT. Because this may result in the AITR occupying positions behind or to one side of the AISUT, collisions generally occur during turns, not during straight movement. However, this is possible, if the AISUT selects to move in the direction of the AITR, which would then have a head-on-collision trajectory with the AISUT. The AITR used for this work moves at the same speed as the AISUT; however, while the prospective collision scenarios would differ somewhat for a faster moving AITR, this would not substantially change the premise of the work. This is also a topic for future exploration.

7. Experimental Results

Four experimental conditions are considered: a single AI actor in two dimensions, multiple AI actors in two-dimensions, a single AI actor in three dimensions, and multiple AI actors in three dimensions. Each is now discussed.

7.1. Single Actor, Two-Dimensional

The base condition, a two-dimensional test with only one AI tester, demonstrates the utility of using an AITR to validate an AISUT. In [32] data was collected regarding the efficiency of using human testers *versus* the AITR. This showed that a single 500 move test required, on average, two minutes to conduct manually. In this same configuration (which used a slightly older version of the interface, lacking controls relevant to three dimensions on an alternate computer to the one used to run the simulations presented herein) 5,750 automated tests of 500 moves each (a total of 2,875,000 moves) required one minute to run. Table 1 presents data for a set of 2,000 execution, 2,000 move (4,000,000 total moves) test runs with a believed-good AI. Table 2 presents data for the same number of runs with an AI with a known error.

For comparison, based on the speed of the human trials presented in [24], each of these tests would require 266 hours and 40 minutes. The average values of 7:11, for the believed-good AI and 6:59 for the known-error AI are approximately 2,050 times as efficient in terms of test completion as the manual version.

Table 1. Single Actor, two dimensional good AI statistics.

	Good AI Version				
	1	**2**	**3**	**4**	**Average**
Executions	2000	2000	2000	2000	2000
Moves per Execution	2000	2000	2000	2000	2000
Executions with Collision	0	0	0	0	0
Total Collisions	0	0	0	0	0
Average Moves to Collision	N/A	N/A	N/A	N/A	N/A
Run Time	7:00	7:39	7:04	7:01	7:11

Table 2. Single actor, two dimensional known error AI statistics.

	Known Error AI Version				
	1	**2**	**3**	**4**	**Average**
Executions	2000	2000	2000	2000	2000
Moves per Execution	2000	2000	2000	2000	2000
Executions with Collision	1986	1985	1978	1983	1983
Total Collisions	14017	13989	13720	14027	13938.25
Average Moves to Collision	285.37	285.94	291.55	285.16	287.00
Run Time	7:01	6:59	6:57	6:59	6:59

7.2. Multiple Actor, Two-Dimensional

The test routines presented in Section 7.1 were enhanced through the addition of another AI actor. This was tested for another set of test runs with the believed-good AI and known-error AI. Results for these tests are presented in Table 3 and Table 4. The addition of a second AI-controlled actor created a marginal increase in the time required to run the 2000 execution, 2000 move-per-execution tests.

This increased from an average of 7:11 to 7:14 for the known-good AI, a three-second increase. The average time increased from 6:59 to 7:10, an 11-second increase, for the known-error AI. Thus, double the number of AI moves were created for an average minor time increase of seven seconds (about 1/60th, or 1.7%, of the total runtime). Both AIs operated within the same loop structure. Consolidation of redundant tasks between the two AIs, as well as a single rendering process were likely responsible for the enhanced performance (as opposed to taking twice as long as the single AI presented in Section 7.1)

Table 5 presents a single test for a scenario where the presence of the second AI bot was not checked for prior to movement into the space. This demonstrates the utility of the test environment for testing multiple craft configuration scenarios (e.g., a craft with camouflage capabilities). It also shows the value of adding the second craft for creating collisions, as this craft was responsible for 4704 collisions and a collision on approximately 75% of the 2000 test runs.

Table 3. Multiple actor, two-dimensional good AI statistics.

	Good AI Version				
	1	**2**	**3**	**4**	**Average**
Executions	2000	2000	2000	2000	2000
Moves per Execution	2000	2000	2000	2000	2000
Executions with Collision	0	0	0	0	0
Total Collisions	0	0	0	0	0
Average Moves to Collision	N/A	N/A	N/A	N/A	N/A
Run Time	7:08	7:22	7:14	7:12	7:14

Table 4. Multiple actor, two-dimensional known-error AI statistics.

	Known Error AI Version				
	1	**2**	**3**	**4**	**Average**
Executions	2000	2000	2000	2000	2000
Moves per Execution	2000	2000	2000	2000	2000
Executions with Collision	2000	1999	1999	1999	1999.25
Total Collisions	34823	34618	34909	35354	34926
Average Moves to Collision	114.87	115.55	114.58	113.14	114.53
Run Time	7:09	7:09	7:07	7:17	7:10

Table 5. Multiple actor, two-dimensional good AI, checking only the first tester statistics.

	First Bot Check Only
Executions	2000
Moves per Execution	2000
Executions with Collision	1478
Total Collisions	4704
Average Moves to Collision	327
Run Time	7:07

7.3. Single Actor, Three-Dimensional

Another direction of enhancement of the work presented in Section 7.1 is the testing of AI operations in three dimensions. This adds a level of complexity to the AI operations and complicates the human control of the actor (presumed to be a UAV), due to difficulties in perceiving the ideal direction of travel required to intersect the AISUT actor and with close-proximity maneuvering (given the difficulty of visualizing the exact altitude of the craft from subtle differences in color shading, which is lightened to show higher altitude and darkened to show lower altitudes). Alternately, indicating values numerically was tried. However, both created significant increases in the amount of time required for human craft operations. This time increase was attributed to having to calculate the best maneuver, as opposed to arriving at this intuitively, as was possible in two dimensions. Identifying and creating a more effective visualization mechanism is a prospective subject of future work. Table 6 presents data for a believed-good AI (which exhibits a small error in three-dimensional operations, causing an average of two collisions across 4,000,000 moves). Table 7 presents data for an AI with a known error.

Table 6. Single actor, three-dimensional good AI statistics.

	Good AI Version		
	1	**2**	**Average**
Executions	2000	2000	2000
Moves per Execution	2000	2000	2000
Executions with Collision	1	1	1
Total Collisions	3	1	2
Average Moves to Collision	1333333.33	4000000.00	2666666.67
Run Time	7:12	7:01	7:07

Table 7. Single actor, three-dimensional known error AI statistics.

	Known Error AI Version		
	1	**2**	**Average**
Executions	2000	2000	2000
Moves per Execution	2000	2000	2000
Executions with Collision	1997	2000	1998.5
Total Collisions	83342	87079	85210.5
Average Moves to Collision	48.00	45.94	46.97
Run Time	7:00	7:16	7:08

7.4. Multiple Actor, Three-Dimensional

The work presented in Section 7.3 is extended (paralleling Section 7.2's extension of the work in Section 7.1) to incorporate multiple AITR actors. The incorporation of multiple actors exposed an issue in the believed-good collision avoidance routine which caused numerous collisions (however, not as many as with the version with the *a priori* known error). The data from the previously believed-good AITR run of 2,000 executions with 2,000 moves per execution (total of 4,000,000 moves) are presented in Table 8. The data from the 2,000 executions with 2,000 moves per execution (total of 4,000,000 moves) on the known-error version is presented in Table 9.

Table 8. Multiple actor, three-dimensional good AI statistics.

	Good AI Version		
	1	**2**	**Average**
Executions	2000	2000	2000
Moves per Execution	2000	2000	2000
Executions with Collision	1999	2000	1999.5
Total Collisions	86804	83115	84959.5
Average Moves to Collision	46.08	48.13	47.10
Run Time	7:14	7:00	7:07

Table 9. Multiple actor, three-dimensional known error AI statistics.

	Known Error AI Version		
	1	**2**	**Average**
Executions	2000	2000	2000
Moves per Execution	2000	2000	2000
Executions with Collision	1995	2000	1997.5
Total Collisions	188649	190446	189547.5
Average Moves to Collision	21.20	21.00	21.10
Run Time	7:13	7:30	7:22

8. Analysis of Experimental Results

Table 10 and Table 11 present the run times for the two-dimensional and three-dimensional conditions. Comparison indicates that the known-error AITR marginally outperforms (in terms of speed) the believed-good AITR in the two-dimensional conditions. The believed-good AITR marginally outperforms the known-error AITR in the three-dimensional conditions. In each case, the difference is a small fraction (under 3.6%) of total operating time.

The AITR significantly outperforms manual human-performed operation (described in [24]). Presuming that all of the more complex testing (e.g., that described in Sections 7.2, 7.3 and 7.4) could be performed by humans at a simple multiple of the single actor performance time (which actual testing reveals to be a better-than-best-case projection), each represents a significant performance enhancement, as compared to manual testing (Table 12 presents these projected values; Table 13 presents the level of over-performance as a multiple of how many AI moves can be completed in the time of a single manual move). As testing shows that humans perform slower (requiring more than two times the time to maneuver two separate actors) when moving from single actor to multi-actor scenarios and machine performance increases marginally (a three second increase for the believed good AITR and an 11 second increase for the known-error AITR), the comparative value of the AITR approach increases. Perception and control difficulties made the human performance on the three-dimensional conditions unsuitable for comparison; however, the AITR performs comparably for all four conditions.

Table 10. Comparison of run times between two-dimensional conditions.

	Single Actor		**Multi-Actor**	
	Good	**Known Error**	**Good**	**Known Error**
Test 1	7:00	7:01	7:08	7:09
Test 2	7:39	6:59	7:22	7:09
Test 3	7:04	6:57	7:14	7:07
Test 4	7:01	6:59	7:12	7:17
Average	7:11	6:59	7:14	7:10

Table 11. Comparison of run times between three-dimensional conditions.

	Single Actor		**Multi-Actor**	
	Good	**Known Error**	**Good**	**Known Error**
Test 1	7:12	7:00	7:14	7:13
Test 2	7:01	7:16	7:00	7:30
Average	7:07	7:08	7:07	7:22

Table 12. Projected human time required to finish a 2,000 execution, 2,000 move-per-execution run.

	Single Actor	**Multi-Actor**
2-Dimensional	16,000	32,000
3-Dimensional	24,000	48,000

Table 13. Projected multiple of over-performance of manual trial by AITR.

	Single Actor	**Multi-Actor**
2-Dimensional	2,258.82	4,444.44
3-Dimensional	3,368.42	6,628.31

9. Conclusions

The work presented herein demonstrates that an artificial intelligence test case producer can be utilized to effectively test artificial intelligence systems for both surface (two-dimensional) and airborne (three-dimensional) robots. The testing demonstrated the ability of the AITR to identify bugs in an AISUT. It also demonstrates the utility of using the AITR instead of manual testing. The AITR is able to perform thousands of maneuvers in the same amount of time as a human can perform a single maneuver. This allows significantly more testing to be performed, more thoroughly validating the AISUT. Moreover, the demonstrated capability of the AITR to generate near-optimal routes to the AISUT actor allows significantly enhanced performance, as compared with human actors (in addition to speed benefits). A key limitation, the fact that the AISUT and AITR actors move at the same speed dramatically limited the level of strategy and adaptation that could be incorporated in the testing routine. Given that the AITR could not accurately predict where the AISUT's actors would be at any future point, it is unable to position its actor or actors to try particular approach configurations. A multitude of these prospective occur by happenstance. Human testers encountered the same issue and human intuition did not fare better than the AI at positioning the AITR's actors. In fact, humans underperformed due to generating less-than-optimal routes to a given target point.

10. Future Work

Future work will involve additional refinement of using artificial intelligence to test the two artificial intelligence systems discussed. This work will begin with changing the movement speed of the AITR to facilitate the incorporation of strategic elements and a system that refines this strategy. It will also involve testing applications of the AITR to different types of AISUTs. Finally, refinement of the three-dimensional display system will be conducted in an attempt to enhance the performance of human testers in the three-dimensional scenarios.

Acknowledgments

The feedback provided by Dr. Hassan Reza is gratefully acknowledged.

Conflict of Interest

The authors declare no conflict of interest.

References and Notes

1. Halawani, S. Safety Issues of computer Failure. Technical Report. Available online: http://amubaraki.kau.edu.sa/Files/830/Researches/55979_26288.doc (accessed on December 7, 2012).
2. AdiSrikanth; Kulkarni, N.J.; Naveen, K.V.; Singh, P.; Srivastava, P.R. Test Case Optimization Using Artificial Bee Colony Algorithm. In Proceedings of ACC 2011, Part III, CCIS 192, Springer Verlag: New York, NY, 2011; pp. 570–579.
3. Mondada, F.; Floreano, D. Evolution of neural control structures: some experiments on mobile robots. *Robotics Auton. Syst.* **1995**, *16*, 183–195.
4. Felgenbaum, E.A. The Art of Artificial Intelligence, Stanford University Technical Report, STAN-CS-77-621, Stanford University: Cambridge, MA, 1977.
5. Chandrasekaran, B. On Evaluating AI Systems for Medical Diagnosis. *The AI Magazine* **1983**, *48*, 34–37.
6. Cholewinski, P.; Marek, V.W.; Mikitiuk, A.; Truszczynski, M. Computing with Default Logic. *Artif. Intell.* **1999**, *112*, 105–146.
7. Brooks, R.A. *Artificial Life and Real Robots*. MIT Artificial Intelligence Laboratory: Cambridge, MA, USA (in press).
8. Boooks, R.A. Elephants Don't Play Chess. In *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*; Maes, P., Ed.; MIT Press: Cambridge, MA, USA,1990.
9. Brooks, R.A. Intelligence Without Reason. In Proceedings of the International Joint Conferences on Artificial Intelligence; Morgan Kaufmann Publishers: San Francisco, CA, USA, 1991; pp. 569–595.
10. Brooks, R.A. New Approaches to Robotics. *Science* **1991**, *253*, 1227–1232.
11. Billings, D.; Davidson, A.; Schaeffer, J.; Szafron, D. The Challenge of Poker. *Artif. Intell.* **2002**, *134*, 201–240.
12. Dai, P.; Mausam; Weld, D.S. Artificial Intelligence for Artificial Artificial Intelligence. In Proceedings of the 25th AAAI Conference on Artificial Intelligence; AAAI Press: Palo Alto, CA, USA, 2011; pp. 1153–1159.
13. Pitchforth, J.; Mengersen, K. A Proposed Validation Framework for Expert Elicited Bayesian Networks. *Expert Syst. Appl.* **2012**, *40*, 162–167.
14. Wotawa, F.; Nica, S.; Nica, M. Debugging and test case generation using constraints and mutations. In Proceedings of the 9th Workshop on Intelligent Solutions in Embedded Systems (WISES); IEEE: New York, NY, USA, 2011; pp. 95–100.
15. Suri, B.; Singhal, S. Analyzing Test Case Selection & Prioritization using ACO. *ACM SIGSOFT Softw. Eng. Notes* **2011**, *36*, 1–5.
16. Harman, M. The role of Artificial Intelligence in Software Engineering. In Proceedings of the 1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE); IEEE: New York, NY, USA, 2012; pp.1–6.

17. Pop, C.B.; Chifu, V.R.; Salomie, I.; Baico, R.B.; Dinsoreanu, M; Copil, G. A Hybrid Firelfy–Inspired Approach for Optimal Semantic Web Service Composition. *Scalable Comput.: Pract. Exp.* **2011**, *12*, 363–369.
18. Shah-Hosseini, H. Problem Solving by Intelligent Water Drops. In Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, 25–28 September 2007; IEEE: New York, NY, USA, 2007; pp. 3226–3231.
19. Duan, H.; Liu, S.; Wu, J. Novel Intelligent Water Drops Optimization Approach to Single UCAV Smooth Trajectory Planning. *Aerosp. Sci. Technol.* **2009**, *13*, 442–449.
20. Gendreau, M.; Hertz, A.; Laporte, G. A Tabu Search Heuristic for the Vehicle Routing Problem. *Manag. Sci.* **1994**, *40*, 1276–1290.
21. Glover, F. Heuristic for Integer Programming Using Surrogate Constraints. *Decis. Sci.* **1977**, *8*, 156–166.
22. Glover, F. Tabu Search: A Tutorial. *Interfaces* **1990**, *20*, 74–94.
23. Yang, X.; Deb, S. Cuckoo Search via Levy Flights. In Proceedings of the World Congress on Nature and Biologically Inspired Computing, India, 2009; IEEE: New York, NY, USA, 2009; pp. 210–214.
24. Walton, S.; Hassan, O.; Morgan, K; Brown, M.R. Modified Cuckoo Search: A New Gradient Free Optimisation Algorithm. *Chaos, Solitons & Fractals* **2011**, *44*, 710–718.
25. Bulatovic, R.R.; Dordevic, S.R.; Dordevic, V.S. Cuckoo Search Algorithm: A Metaheuristic Approach to Solving the Problem of Optimum Synthesis of a Six–Bar Double Dwell Linkage. *Mech. Mach. Theory* **2013**, *61*, 1–13.
26. Gandomi, A.H.; Yang, X.; Alavi, A.H. Cuckoo Search Algorithm: a Metaheuristic Approach to Solve Structural Optimization Problems. *Eng. Comput.* **2013**, *29*, 17–35.
27. Huckle, T., Collection of Software Bugs. Institut für Informatik TU München: Munich, Germany. Available online: http://www5.in.tum.de/~huckle/bugse.html (accessed on 7 December 2012).
28. Jet Propulsion Laboratory. Mars Climate Orbiter. Jet Propulsion Laboratory: Pasadena, CA. Available online: http://mars.jpl.nasa.gov/msp98/orbiter/.
29. Dershowitz, N. Software Horror Stories. Tel Aviv University School of Computer Science: Tel Aviv, Israel. Available online: http://www.cs.tau.ac.il/~nachumd/verify/horror.html (accessed on 7 December 2012).
30. Leveson, N. Medical Devices: The Therac–25. Massachusets Institute of Technology: Cambridge, MA. Available online: http://sunnyday.mit.edu/papers/therac.pdf (accessed on 7 December 2012).
31. Jorgensen, P. *Software Testing: A Craftsman's Approach*; Auerbach Publications: Boca Raton, FL, USA, 2008; pp. 7–8.
32. Huber, J.; Straub, J. Human Proximity Operations System Test Case Validation. In Proceedings of the 2013 IEEE Aerospace Conference, Big Sky, MT, USA, 2–9 March 2013; IEEE: New York, NY, 2013 (in press).