TEST-DRIVEN SIMULATION MODELLING: A CASE STUDY USING AGENT-BASED MARITIME SEARCH-OPERATION SIMULATION

Bhakti Stephan Onggo *)

Department of Management Science
Lancaster University Management School
Lancaster University
Lancaster, LA1 4YX, UNITED KINGDOM

Mumtaz Karatas

Department of Industrial Engineering
Turkish Naval Academy
Tuzla, Istanbul, 34942, TURKEY

*) corresponding author (Email: s.onggo@lancaster.ac.uk, Phone: +44 1524 594237, Fax: +44 1524 592060)

ABSTRACT

Model verification and validation (V&V) is one of the most important activities in simulation modelling. Model validation is especially challenging for agent-based simulation (ABS). Techniques that can help to improve V&V in simulation modelling are needed. This paper proposes a V&V technique called Test-Driven Simulation Modelling (TDSM) which applies techniques from Test-Driven Development in software engineering to simulation modelling. The main principle in TDSM is that a unit test for a simulation model has to be specified before the simulation model is implemented. Hence, TDSM explicitly embeds V&V in simulation modelling. We use a case study in maritime search operations to demonstrate how TDSM can be used in practice. Maritime search operations (and search operations in general) are one of the classic applications of Operational Research (OR). Hence, we can use analytical models from the vast search theory literature for unit tests in TDSM. The results show that TDSM is a useful technique in the verification and validation of simulation models, especially ABS models. This paper also shows that ABS can offer an alternative modelling approach in the analysis of maritime search operations.

1 INTRODUCTION

Agent-Based Simulation (ABS) has become one of the commonly used tools to model and understand complex and nonlinear systems (North and Macal 2007). ABS provides a controlled environment for systematic experimentation using a simulation model that is formed from a set of interacting agents. There is no consensus on the definition of an agent in the ABS literature (Macal and North 2010). Instead, we have observed a spectrum of complexity in its definition. At one extreme, an ABS model is formed by a set of agents with a set of simple attributes (such as speed and detection range) and simple behaviours (such as move and rescue). At the other extreme, an ABS model can be composed of a set of agents with complex attributes (such as memory and bounded rationality) and complex abilities (such as planning and learning). However, most researchers agree that an agent is an autonomous entity (i.e. it makes independent decisions without any central control), has a set of objectives and interacts with other agents and its environment.

One of the most important activities in ABS is model validation. There are similarities between ABS and Discrete-Event Simulation (DES). Both are able to represent stochastic dynamic systems and can track individuals' states throughout their lifecycles in the model. Hence, a number of validation techniques proposed for DES are also suitable for ABS, such as face validity, operational validity, white-box validation and black-box validation. Kleijnen (1995), Balci (1995) and Sargent (2013) provide a list of validation techniques in the context of stochastic dynamic simulation which are applicable to DES and ABS. All good simulation textbooks have at least one chapter that discusses validation techniques. Law (2014, chapter 5) and Banks et al. (2010, chapter 10) discuss various techniques, such as increasing the face validity of a model (e.g. by involving domain experts and model users), checking the validity of model assumptions, validating the components of a model, comparing the behaviour of a model with a real system (or something that can represent a real system if a system does not exist). Pidd (2004, pp. 233–246) divides validation techniques into white-box and

black-box validation. White-box validation techniques aim to ensure that the internal working of a simulation model can be justified. Black-box validation techniques compare the output of a simulation model with the output of a benchmark (such as a real-world system similar to the system being modelled or an analytic model).

Despite the similarities, some researchers (e.g. Ormerod and Rosewell 2006, Windrum et al. 2007, Klugl 2008, Duong 2010) have noted that model validation in ABS is especially challenging and identified a number of common challenges. First, we often need to represent the behaviours of agents and the interactions between agents using a set of logical rules. It is challenging to extract this information from social and intelligent agents, such as people and organisations, especially if the agents do not want to be exposed (such as pirates or human traffickers). Furthermore, real-world agents are often heterogeneous. Hence, it is challenging to validate whether the rules used in an ABS model represent the rules used by most real-world agents and whether we have represented the heterogeneity of real-world agents correctly. Secondly, there is a need to validate ABS models at various levels (agent/micro level, system/macro level and intermediate/meso levels). It is challenging to validate behaviour at the system level based solely on knowledge of the behaviours of individual agents. For example, Duong (2010) explains that emergence does not exist before a simulation is run (it might not even exist in the modeller's mind); hence, techniques such as structured walkthrough to analyse emergence from a model without running it would be virtually impossible. Even if we can generate traces during a simulation run, it is still a great challenge to explain how behaviour at a lower level can cause emergence at a higher level. Finally, an ABS model often requires high-fidelity data. Although the collection of high-fidelity data has become very common, qualitative behavioural data from heterogeneous agents in a population are rarely available. Hence, empirical validation may not be possible. The difficulty in validating an ABS model is reflected somewhat in the survey done by Heath et al. (2009). They surveyed 279 research articles and found that only 35 per cent of the models were validated both conceptually (white box) and operationally (black box). Windrum et al. (2007) conduct an interesting discussion about the methodological issues surrounding the empirical validation of ABS. Hence, the challenge is not simply one of data availability, it is also methodological. Given these challenges, an automated tool that can help modellers to validate ABS models is useful.

The objective of this paper is to propose a technique called Test-Driven Simulation Modelling (TDSM) for the verification and validation (V&V) of an ABS model. TDSM's basic principle is that a test case for a simulation model has to be specified before the simulation model is implemented. Hence, the main advantage of TDSM is that the V&V process is explicitly embedded in the simulation-model development process. Modellers are forced to think about how their model is going to be verified and validated, even before they begin to develop it. The second advantage is that TDSM can be implemented using various unit-testing tools, or incorporated into a framework such as the one proposed by Gurcan et al. (2013), Niazzi et al. (2009) and Thiele et al. (2014). To demonstrate how the proposed validation technique can be used in practice, we will show a few examples in which parts of the agent-based simulation software called MASSIM (MAritime Search SIMulation) are verified and validated using TDSM.

The analysis of search operations is a classic part of Operational Research (OR) and has a long history commencing with military operations during the Second World War. Since then, the applications and tools have continued to be mostly nautical, e.g. military assets looking for enemy submarines or maritime pirates, coastguards conducting search-and-rescue (SAR) operations, and patrol boats protecting ports or high-value assets. Since a search operation requires considerable time and effort, it needs to be planned and conducted efficiently. A tool that can help to analyse the expected performance of a search-operation strategy is very useful. Tools that use ABS in the maritime search domain and sea-patrol operations are surprisingly scarce in comparison to other OR techniques, as confirmed by Davidsson et al. (2005) and Vaněk et al. (2013). One of the main reasons is the difficulty in validating the ABS model. This is unfortunate because ABS has unique characteristics (such as the explicit specification of individuals' behaviours and their interactions) which offer an alternative modelling approach. Hence, the findings from this paper can also contribute to the application of ABS in maritime-search operations modelling.

This paper is organized as follows. In Section 2, we review related work in TDSM and the application of ABS in maritime search operations. We explain our generic tool for maritime search

operations, called MASSIM, in Section 3. Section 4 discusses how TDSM is used to validate a number of scenarios in MASSIM. Finally, we present our conclusions and recommendations for future work in Section 5.

2 RELATED WORK

2.1 Test-driven simulation modelling

The idea behind Test-Driven Simulation Modelling (TDSM) comes from Test-Driven Software Development, known simply as Test-Driven Development (TDD) in software engineering. There are a number of definitions and aliases for TDD (Janzen and Saiedian 2005). However, the main principle is that a test case for computer code must be created before the computer code is developed (Beck 2003). The test case is implemented as a unit test. Unit testing is a method in software engineering that is used to test an individual unit of computer code. The individual unit can be a function, a procedure or a class. Because the code does not exist when the unit test is created, the first test will always fail. A programmer will then refine the code until it passes the unit test. Subsequently, a new unit test is created and the process is repeated. At some point, the programmer may need to refactor the code. Refactoring changes the internal structure of the code without changing its observable behaviour (i.e. the code still carries out the same function that it did before). The objective of refactoring is to make the code more readable and easier to maintain. These steps are commonly known as Red-Green-Refactor, which signifies the cycle of creating unit tests that fail, writing code that passes tests and refactoring the code. TDD is a practice that is commonly used with other practices in a software development process.

TDD is suitable for a software development process that is iterative, incremental and evolutionary (Janzen and Saiedian 2005). A development process is iterative when it involves the repetition of development tasks, usually with an incremental set of requirements. Each increment in the requirements results in a new software release. An evolutionary development process uses feedback from previous iterations to improve the software and guide future iterations. Simulation modelling is also an iterative, incremental and evolutionary process. Simulation modellers often revisit stages in a simulation modelling process (e.g. conceptual modelling, computer implementation) iteratively. In each iteration, specifications may be modified and new specifications may be added based on new information or feedback from relevant stakeholders. Hence, the simulation-modelling process is likely to benefit from TDD.

Collier and Ozik (2013) conducted the first exploratory study to investigate how TDD and unit testing could be used to verify an ABS model written in the Repast simulation library. They argued that by focusing on writing small unit tests, complex simulation code could be decomposed into smaller and more manageable components. Asta et al. (2014) followed the same approach and applied it to a discrete-event simulation (DES) model written using AnyLogic. Unlike Repast, AnyLogic is a visual interactive modelling software. Hence, the unit-test code was not written directly as simulation source code. Instead, they wrote each unit test using a user-defined function in AnyLogic to verify a component in the model. Onggo et al. (2014) proposed TDSM as an approach for simulation-model verification and validation. TDSM can be applied to both DES and ABS. TDSM makes use of two unit-test suites, a verification suite and a validation suite. The verification suite contains a set of test cases to check the correctness of a simulation model against its conceptual model. Each test case represents one scenario that checks the correctness of the model. For example, one scenario may test the correctness of the distance travelled by a ship, i.e. at a given speed v and travel time t, the distance between the original location of a ship and its new one is vt. The validation suite contains a set of unit-test cases to check the validity of a simulation model. This is closely linked to black-box validation. Each test case represents a scenario that compares the output of a simulation model against what is expected. For example, in this paper, we use an analytical model to provide estimations of or bounds on certain simulation outputs. These numbers are then used in validation-suite tests. Similar to Collier and Ozik (2013) and Onggo et al. (2014), Gurcan et al. (2013) advocate the use of an automatic testing tool for ABS model verification and validation. However, they adopted a different approach. Instead of using TDD, they proposed a generic testing framework for ABS models. The framework was implemented in Java to work with Repast and JUnit. However, the authors acknowledge that the

next step in their work would be to integrate their framework with TDD. Research into the possible application of TDD in simulation is relatively new. Hence, the number of articles published on this topic is low.

2.2 Application of ABS in maritime search operations

Existing applications of ABS for maritime search operations mainly focus on maritime patrol simulations to deter pirate attacks (Decraene et al. 2010a, Vaněk et al. 2010, Jakob et al. 2010, Bruzzone et al. 2011, Jakob et al. 2011, Tsilis 2011, Jakob et al. 2012, Vaněk et al. 2012, Vaněk et al. 2013, Marchione et al. 2014, Dabrowski and De Villiers 2015, Varol and Gunal 2015), port-protection simulation (Leathrum et al. 2009, Shieh et al. 2012, Harris et al. 2013), the simulation of traffic in ports and coastal waters (Hasegawa 2004), combat simulation (Champagne et al. 2003, Price 2003, Cioppa et al. 2004, Hill et al. 2004, Champagne 2004, Hill et al. 2006, Decraene et al. 2010b, Xing et al. 2013), surface surveillance simulations (Steele 2004) and force protection simulations (Harney 2003, Akbori 2004, Walton et al. 2005, Sullivan 2006, Ng 2007). It is clear that most of the applications are for maritime security and military operations.

In maritime security applications, ABS is used to analyse strategies to protect merchant vessels from pirates. For example, Decraene et al. (2010a) proposed utilizing ABS to investigate the requirements for defending a large commercial vessel against hijacking by pirates. In their piracy simulation model, the authors applied data farming to generate a range of simulation model variants and an Automated Red Teaming (ART) technique to reveal a commercial vessel's critical vulnerabilities to pirates. Jakob et al. (2010) developed a test bed which combines simulated vessel operation with real-world data on maritime activity. They used this test bed to prototype and evaluate agent-based methods for fighting maritime piracy. However, the authors do not explicitly state the techniques they used to V&V the model. In his study, Tsilis (2011) used ABS to model a group of merchant ships travelling under escort by a warship. Based on simulation results, he inferred that the success of a counter-piracy escort mission basically depends on parameters including the numbers and speeds of pirates and merchant ships and the position and detection range of the warship. Vaněk et al. (2010) modelled the problem of a mobile agent trying to cross an area patrolled by a mobile adversary as a two-player zero-sum game (termed *transit game*). They tested their model on a real-world case of ship transit through areas affected by piracy in the Gulf of Aden and validated and evaluated the effectiveness of the game-theoretic approach by employing an ABS of maritime traffic. Vaněk et al. (2012) and Vaněk et al. (2013) developed an ABS model of maritime traffic that explicitly modelled pirate activities and piracy countermeasures in the Gulf of Aden. They used the model to analyse the design of a new transit-corridor system in the Indian Ocean. In another publication, they combined the ABS model with an optimization model to investigate the counter-measure configurations that yielded the best trade-off between security and cost in maritime transportation (Jakob et al. 2012). Marchione et al. (2014) presented an ABS of dynamic patterns of maritime piracy in the Gulf of Aden to estimate the number of pirates and their area of action. Varol and Gunal (2015) proposed a hybrid DES and ABS model to simulate hypothetical scenarios in the Gulf of Aden to better understand the cause and effect relationship between naval resource allocation and piracy prevention. The typical agents used in maritime-transportation security applications are merchant vessels, navy vessels and pirate vessels. The software used by researchers includes AgentC (Vaněk et al. 2010, Jakob et al. 2011, Jakob et al. 2012, Vaněk et al., 2013), PANOPEA (Bruzzone et al. 2011), MANA (Walton et al. 2005, Decraene et al. 2010a, 2010b, Tsilis 2011, Xing et al. 2013) and SharpSim (Varol and Gunal 2015). PANOPEA uses DES but it has intelligent agent components embedded in the model.

There has been an increase in efforts to apply simulation to port protection and port traffic management since 9/11, for both training and planning purposes. For example, Advanced Disaster Management Simulator (ADMS) is a simulation tool developed to train port security personnel in how to respond to a terrorist attack on a seaport (McCard 2015). Shieh et al. (2012), used simulation to analyse the robustness of PROTECT, a game-theoretic system deployed by the United States Coast Guard (USCG) in the port of Boston. The main objective is to quantify the uncertainty that might arise in the real world. Harris et al. (2013) used Automated Vulnerability Evaluation on Risks of Terrorism (AVERT) software to implement ABS for the effective utilization of defensive security

resources in US ports. They provided a quantitative basis for recommending various patrol patterns and defensive measures that could decrease risk and enhance deterrence strategies.

Military applications vary. Cioppa et al. (2004) noted that there was increasing interest in the use of ABS in the US Department of Defense. They highlight three examples of ABS military applications. One of these is the study of how unmanned surface vehicles could be used in force-protection missions. A simulation of the Bay of Biscay U-boat campaign has been reported several times (e.g. Champagne et al. 2003, Champagne 2004, Hill et al. 2004, Hill et al. 2006). Champagne et al. (2003) studied the emergent behaviours of combatants and the effectiveness of search patterns during the campaign. Price (2003) and Hill et al. (2004) incorporated game theory into the behaviours of agents in the model. Champagne (2004) also proposed a statistical validation methodology based on re-sampling historical outcomes, which allows comparison between a simulation and historic operation. Using this methodology he showed that the Bay of Biscay ABS is a good representation of the real-world operation. The US Navy also utilized ABS for a Naval Simulation System which was developed to support network-centric fleet battle exercises (Metron Incorporated 2015). DeStefano (2004) utilized ABS to create an executable model of a weapons system built in an agent-based combat model "System Effectiveness Analysis Simulation (SEAS)". Decraene et al. (2010b) extended ART to broaden the range of evolvable simulation model parameters and considered a maritime anchorage protection scenario where individual trajectories of belligerent vessels are evolved to break Blue. To illustrate the potential benefits of a simulation screening procedure, Xing et al. (2013) applied it to maritime escort operations in the Strait of Gibraltar.

At the US Naval Postgraduate School (NPS), a series of studies have been conducted to model and simulate force protection operations. For example, Harney (2003) used ABS to develop a prototypical planning tool for Anti-Terrorism and Force Protection for Navy ships. Sullivan (2006) expanded Harney's (2003) work by including a capability for testing force protection measures in multiple scenarios. Akbori (2004) developed an Anti-Submarine Warfare (ASW) screen design simulation to aid ASW commanders in configuring an ASW screen. Walton et al. (2005) studied ways of preventing successful small-boat attacks against larger high-value commercial ships through the utilization of ABS. As a case study, they analysed the protection of merchant ships transiting the Straits of Malacca and extend the results of their analysis to other ports and their local waterways. Ng (2007) presented models of asymmetric threats in maritime security and used ABS to provide complex adaptive behaviours for threats. In his thesis, Steele (2004) used ABS to explore alternative configurations of the prototype and operational uses of unmanned surface vehicles (USVs) for three scenarios: maritime interdiction, surveillance and reconnaissance, and force protection. He provided operational and tactical insights into how to use USVs in maritime missions.

Some of the models mentioned above were calibrated using real-world data. For example, trajectory data for vessels (from satellite or self-reporting systems) were used to calibrate the paths taken by vessels (Vaněk et al. 2013). Similarly, Marchione et al. (2014) and Dabrowski and De Villiers (2015) used empirical observations concerning the volume of vessels sailing through the Gulf of Aden. In the absence of data, Marchione et al. (2014) used a Genetic Algorithm (GA) to estimate parameters to calibrate some of the model variables. In the case of the Bay of Biscay U-boat campaign simulation, the authors used historical data from the war. However, the behaviour of hostile or non-cooperative agents such as pirate vessels is difficult to find, and in many cases unavailable. Hence, the calibration of such agents using empirical data is limited. Researchers can use an analytical approach to model the behaviour of such agents so that the behaviour shown by the simulation model should match the underlying analytical model. However, consistent with the findings of Heath et al. (2009), model validation is hardly discussed in the literature. Validation has mainly been done using a qualitative approach, such as face validity and an expert's opinion (e.g. Vaněk et al. 2013, DeStefano 2004, Varol and Gunal 2015). Hence, it is clear that more research is needed on the validation of agent-based maritime search-operation models.

3 MARITIME SEARCH SIMULATION (MASSIM)

This section explains MASSIM, a generic ABS tool for maritime search operations that we have developed using Repast (North et al. 2013). First, we will explain how we abstract maritime search

operations in MASSIM. This is followed by a detailed description of the model's structure and parameters.

3.1 Characteristics of maritime search operations

Figure 1 shows the classification of a maritime search-operation problem. Each box represents an alternative for its associated search problem element. Current version of MASSIM supports the features indicated with solid arrows. Dashed arrows indicate the features MASSIM does not support yet, e.g. currently MASSIM cannot handle a search operation in an expanding area conducted with probabilistic sensors.

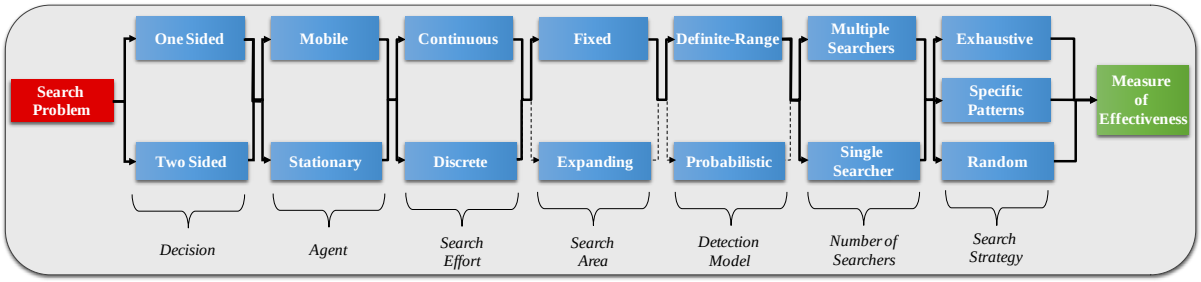


Figure 1: Search problem classification used in MASSIM.

From an ABS perspective, all search problems have two agents in common: a target (in a broad sense, something being searched for), and a searcher (Nunn 1981). An operation can be classified as a two-sided problem when both target(s) and searcher(s) are active and behave in an intelligent way or a one-sided problem when either target(s) or searcher(s) is active and intelligent. The level of intelligence can be as simple as conducting a specific search pattern or as complex as learning from past experience. The agents can be stationary or moving. The search efforts in a discrete search operation are made at a number of discrete locations (e.g. a helicopter dips sonar at a location, pulls it and then flies to another location, and repeats the process for a number of discrete locations). In a continuous search operation, search efforts are carried out continuously along a certain path (e.g. a search conducted by a ship that carries hull-mounted sonar as it moves from one location to another). A logical extension of area-search problems involves searching an expanding area, as studied by Coggins (1971) and Washburn (1980). As opposed to searching a fixed area, in an expanding area search problem, the area where the target is likely to expand over time. One such example is the Flaming Datum Problem (FDP) defined by Washburn and Hohzaki (2001). FDP deals with relocating an enemy submarine that is fleeing after momentarily revealing its position. The detection function used during an operation is called definite range (also known as a "cookie cutter") when a target is detected whenever it is within detection range. In a probabilistic detection function (such as polynomial, exponential and cubic attenuation models), the detection probability is a function of the distance between searcher and target. The analysis of a search operation can consider a single-searcher strategy or a multiple-searchers strategy. The main objective is to analyse the effectiveness of a search strategy in a number of scenarios. Multiple-searcher operations are especially useful in the analysis of the effectiveness of a collaborative search strategy where a large area of interest is divided into sectors or sub-responsibility areas for each searcher. Finally, we can evaluate various search algorithms that can be used in a search operation, such as exhaustive search, random search and specific patterns (e.g. parallel search, expanding square search).

3.2 MASSIM model

MASSIM has been designed to support the analysis of a number of search operation settings, as shown in Figure 1. At the time of writing, MASSIM can support one- or two-sided problems, stationary/ moving agents, discrete/ continuous search efforts, fixed-area search problems, a definite-range detection model, single/ multiple searchers and a number of search patterns including random

search and exhaustive search. The ABS model in MASSIM is formed by two agent types: searcher and target. A searcher's main objective is to detect a target. The key properties of a searcher are:

- Search strategy: This defines the movement algorithm for the searcher;
- Current location: This defines the location of the searcher;
- Search area: This defines the location and size of the searcher's search area;
- Speed: This defines the speed of the searcher;
- Detection: This defines the searcher's detection model and detection range;
- Performance statistics: These are used to collect performance statistics of the searcher.

The flowchart that represents the behaviour of a searcher is shown in Figure 2. The simulation time is advanced using fixed increments (Δt). In each step, the flowchart is executed. First, the searcher will move to a new location depending on the search strategy, speed and Δt. Next, the searcher will see if there is any target within its detection range. This is done by checking all target objects within its detection radius, and depending on the detection model a target may be flagged as non-active (i.e. has been detected). A flag is needed to avoid the double counting of performance statistics (an alternative would be to remove the detected target from the simulation). When a target is detected, counters, such as the number of targets detected, are updated. Finally, performance statistics, such as the total number of targets detected and the total number of successful search efforts, are updated.

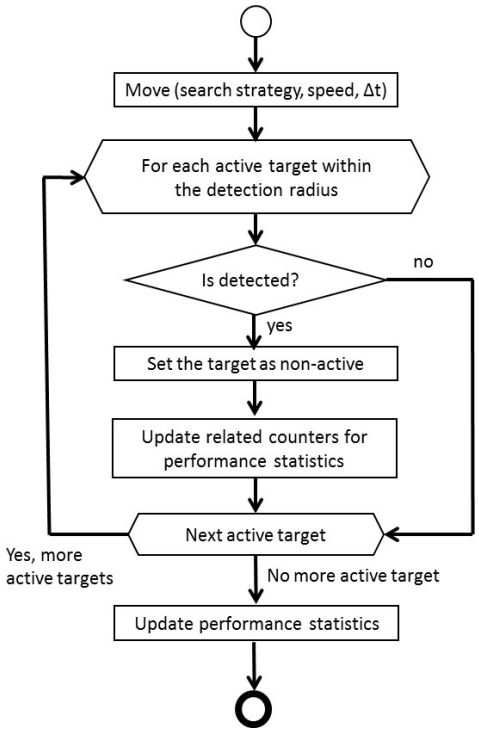


Figure 2: The behaviour of a searcher.

Based on its behaviour, a moving target can be classified into one of three groups: a cooperative target that wishes to be detected by a searcher (e.g. the victims in a SAR operation), a non-cooperative or evading target that wishes to hide or escape from the searcher (e.g. a refugee trying to reach his/her destination without being detected by the coastguard), and a non-cooperative target that wishes to be as close as possible to a searcher without being detected (e.g. a hostile submarine trying to approach surface ships to within its effective torpedo range). The key properties of a target are:

- Movement strategy: This defines the movement algorithm for the target;
- Current location: This defines the location of the target;
- Search area: This is an artefact that is needed to count the frequency and measure the time that the target spends inside a search area;

- Speed: This defines the speed of the target;
- Counter-Detection: This defines the target's counter-detection model and counter-detection range;
- Statistics: These are used to collect statistics related to the target.

The behaviour of a target is shown in Figure 3. First, if the target has not been detected (i.e. active) and its objective has not been met (e.g. for a refugee boat whether it has reached a coastal area, for a victim in an accident whether s/he has been detected by a rescue boat), the target will move to a new location depending on the movement strategy, speed and Δt. The movement strategy depends on the behavioural category of the target, as mentioned earlier. Next, the target will check if its objective is met. In this case, a flag is set to exclude the target from the simulation. Regardless whether the objective has been met or not, related counters and statistics are updated (e.g. the duration until the objective is met, the frequency and time that the target spends inside a search area).

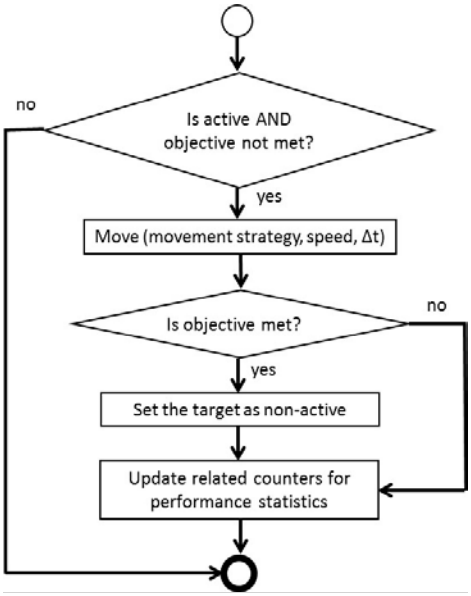


Figure 3: The behaviour of a target.

The interaction between a searcher and a target happens when the target is within the searcher's detection range. The frequency of interactions depends on the number of agents, the agents' movement strategies and whether the target is cooperative or non-cooperative (either evading or approaching).

3.3 Simulation parameters

Figure 4 is a screenshot of MASSIM for a single-searcher operation. In MASSIM each region is modelled as a 2-dimensional plane. All targets lie somewhere in the region of area A and refer to it as the *total area* (shown as the large grey square in Figure 4). For each searcher, we define a search area A' as a sub-region of A (shown as the white square inside the grey square in Figure 4). The searcher and target are denoted by a triangle and a disk, respectively. The detection region of a searcher is denoted by a circle around it. When a definite range sensor is used, a target that lies inside the sensing zone will be detected. In Figure 4, we show one searcher and 100 targets. Two of the targets are within the searcher's detection region.

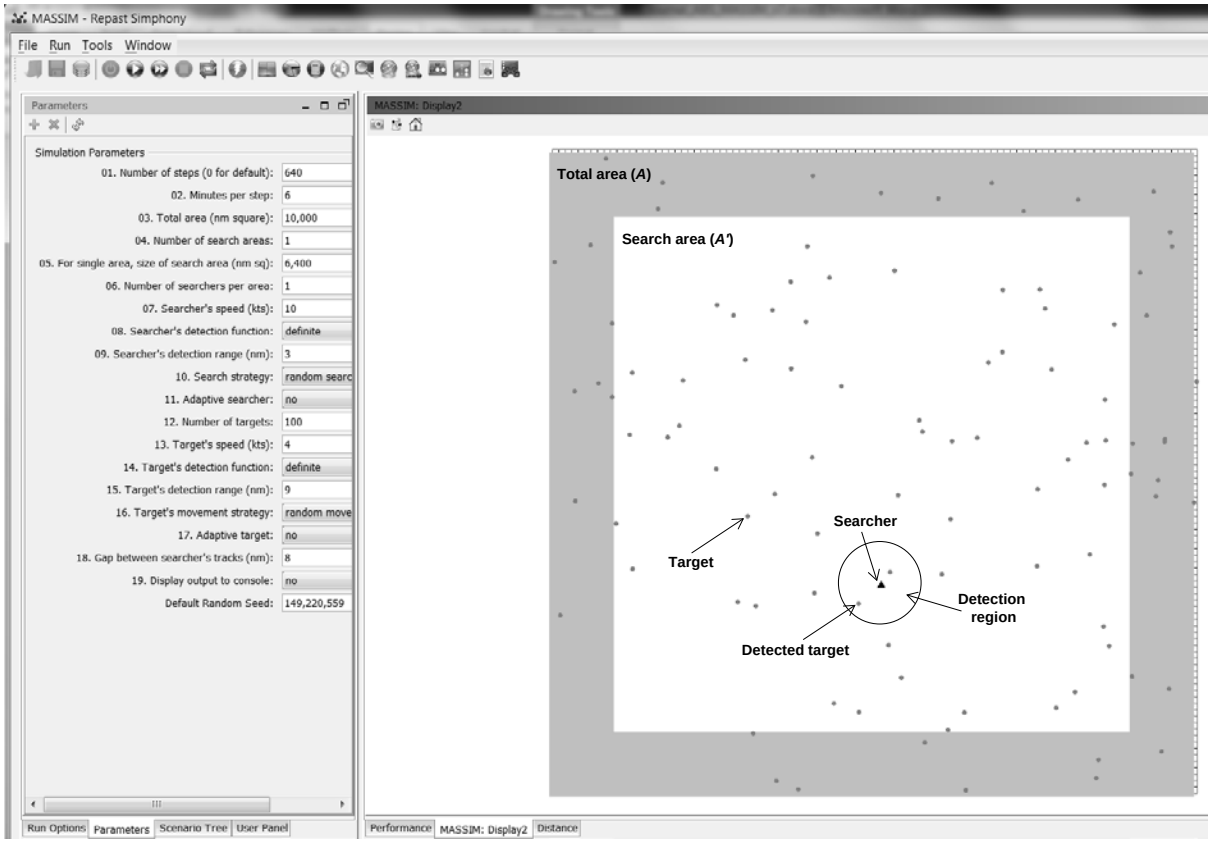


Figure 4: MASSIM screenshot for single-searcher operation.

The simulation parameters can be controlled from the panel on the left in Figure 4. The parameters, their types and possible ranges are summarized in Table 1. The first two parameters are the number of time steps and the duration per time step (in minutes), respectively. For a continuous search operation, a shorter time-step duration leads to a more accurate result. This is because the locations of the agents are updated in every time step. The third parameter defines the total area, A, in nm^2 (nautical miles squared). The area is assumed to be a square to make the simulation visualization easier. The fourth parameter is used to specify the number of equal-size search areas. If we are interested in single-search-area operations, we can specify the size of the search area (parameter 5). Parameter 6 allows us to specify the number of searchers per search area. Hence, the combination of parameters 4 and 6 allows us to analyse single-searcher single-area operations, multiple-searchers single-area operations or multiple-searchers multiple-areas operations. A searcher has a number of attributes (parameters 7 to 11): speed in knots, detection function type (at the moment, it only supports definite range detection function, but it will be extended to include probabilistic functions such as polynomial, exponential and cubic), detection range in nautical miles, search strategy (such as random search, exhaustive search or parallel search), and whether the searcher has the ability to learn. Parameter 12 controls the number of targets in the total area. Each target has the same number of attributes as a searcher (parameters 13 to 17): speed in knots, detection function, detection range, movement strategy (such as stationary or random walk) and whether it has the ability to learn. Parameter 18 is the distance between each track in a parallel/ creeping/ square search in nm.

Table 1: Simulation parameters used in MASSIM. The values in square brackets are the minimum and maximum values of the corresponding parameters.

#	Simulation Parameters	Data Type	Possible Range
1	Number of steps	Integer	$[1,10^4]$ – This parameter controls the search duration (for analysis, ideally the number of steps is set to be the number of steps needed to cover the whole search area using an exhaustive search.)

2	Minutes per step (Δt)	Continuous	[1,6] – This parameter controls the smoothness of the animation, e.g. if we choose 1 minutes per step and the speed of a searcher is v = 10 kts., the searcher will move for 1/6 miles per simulation step.
3	Total area or A (in nm^2)	Integer	$[10^2,10^6]$ – This parameter controls the size of a region of interest.
4	Number of search areas	Integer	[1,25] – This parameter controls the number of searchers.
5	For single area, size of each area or A' (in nm^2)	Integer	$[1,10^6]$ and $A' \leq A$ – This parameter controls the size of a search area (A') within a region of interest (A)
6	Number of searchers per area	Integer	[1,5] – This parameter controls the number of searchers utilized in each search area in an operation.
7	Searcher's speed (in kts.)	Integer	[5,200] – 5 kts. represents an operation in which a searcher (surface ship) conducts a thorough search with low speed and 200 kts. represents a search operation conducted by a maritime patrol aircraft.
8	Searcher's detection function	List	Definite, Probabilistic.
9	Searcher's detection range (in nm)	Integer	[1,40] – Short detection ranges represent visual search in poor environmental conditions, and long detection ranges represent search with optical sensors or radar in favourable environmental conditions.
10	Search strategy	List	Random, Exhaustive, Parallel, Creeping, Expanding Square.
11	Adaptive searcher	Boolean	True, False.
12	Number of targets	Integer	$[1,10^3]$ – This parameter controls the number of targets to be simulated in the region of interest (A).
13	Target's speed (in kts.)	Integer	[0,30] – 0 kts. represents stationary targets and 30 kts. represents high-speed submerged targets.
14	Target's detection function	List	Definite, Probabilistic.
15	Target's detection range (in nm)	Integer	[0,100] – 0 nm. represents stationary targets and 100 nm. represents submerged targets with long detection range capability.
16	Target's movement strategy	List	Stationary, Random, Approaching, Evading.
17	Adaptive target	Boolean	True, False.
18	Gap between searcher's tracks (in nm)	Integer	[1,100] – This is used in parallel, creeping or square search strategies. By default, the gap is set as $2r$ (where r is the detection radius). However, in real operations, we may implement a larger gap especially when we have a budget constraint.
19	Display output to console	Boolean	True, False.

4 TEST-DRIVEN SIMULATION MODELLING

The main principle in TDSM is that the V&V of a simulation model must be specified before the simulation model is implemented. This practice explicitly integrates V&V into the simulation modelling process by forcing modellers to think about V&V before model development starts and to use V&V cases to guide the model development process.

A flowchart for the simulation modelling process with TDSM is shown in Figure 5. The TDSM stages are highlighted in grey. The simulation modelling process starts with the commonly adopted stages of problem structuring, conceptual modelling and input modelling. These stages are followed by the TDSM stage.

In the first part of the TDSM stage, model development is guided by verification cases. The objective of model verification in simulation modelling is to ensure that a conceptual model has been implemented correctly in a simulation model. A model is verified using a number of verification cases. In TDSM, each verification case is implemented as a unit test. Each unit test compares the behaviour of the simulation model against the conceptual model (closely related to white-box testing in software engineering). First, it is necessary to specify verification cases (ideally, in close collaboration with model users and/or domain experts). Afterwards, each verification case is

translated into a unit test. The unit tests form a verification suite. Starting with the first unit test in the verification suite, the model is then incrementally developed until it passes the unit test. When the model passes the test, we move on to the second unit test in the verification suite. This process is repeated until the model passes all unit tests in the verification suite.

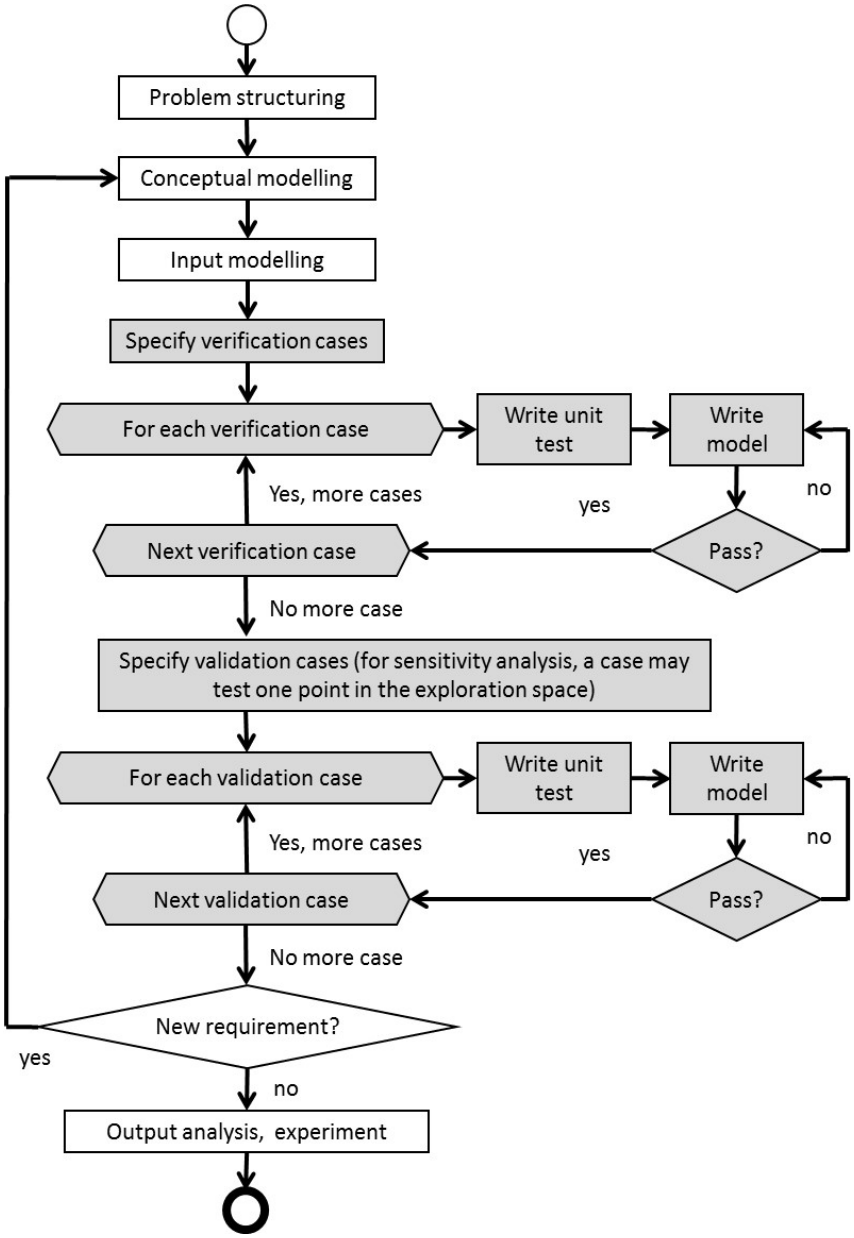


Figure 5: Simulation modelling process with TDSM

Once verification is complete, we then use the same principles for validation. The objective of model validation is to make sure that a simulation model is fit for its purpose. As in model verification, a model is validated using a number of validation cases and each validation case is implemented as a unit test. Each unit test checks the validity of a simulation model by comparing its output to the expected behaviour in the real world (using empirical data or an analytical model or theory if empirical data are not available). Hence, this technique can be applied to both data-driven models and theory-driven models. This comparison is closely linked to black-box testing. It should be noted that model validation often includes sensitivity analysis that explores the behaviour of the model under various combinations of parameter values (within acceptable ranges) as discussed in Thiele et al. (2014). In this case, a validation case tests the behaviour of the model under a specific combination of parameter values. Hence, the number of validation cases grows exponentially with the

number of parameters. For this reason, in addition to better exploration algorithms, Thiele et al. (2014) highlight the importance of an automated tool (such as our tool) for model validation. As shown in Figure 5, we start with the specification of validation cases in close collaboration with model users (and/or domain experts). This is followed by implementing validation cases as unit tests which will be grouped as a validation suite. The model will then be developed incrementally until it passes all unit tests in the validation suite. At this stage the model has been completely validated against the verification and validation cases. At some points during the iteration, it may be necessary to refactor the model to make it more structured, modular and easier to maintain. If there are no new requirements raised by the model users, the model is ready for output analysis and experiments. Otherwise, the process will be repeated. Subsequent activities follow the usual simulation modelling practice.

In cases where the simulation model is likely to be used and modified in the future, the unit tests should not be discarded after the model has been verified and validated. This is because the unit tests can be used to detect if any of the alterations or additions to the model break the existing unit tests as long as the unit tests remain relevant. For example, Figure 6 shows that our model has passed four validation cases, as indicated by the check signs. When we extend or modify the model, we need to make sure that we do not break any of these cases. Of course, if a new requirement makes a unit test no longer relevant, the unit test should be removed from the verification and validation suites.

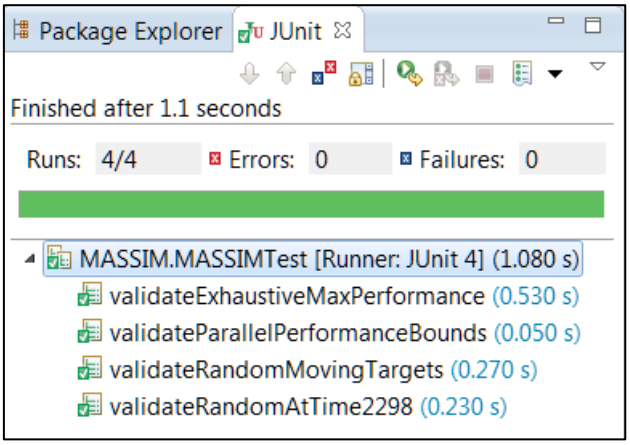


Figure 6: A snapshot of TDSM validation result

Each unit test is essentially carrying out a comparison. During verification, we can compare the behaviour of the model with what is expected based on the conceptual model. To take an example from MASSIM, a searcher moves at speed v to the north from its current location (s_x, s_y). After time period Δt, the new location of the searcher will be $(s_x, s_y + v\Delta t)$. This scenario can be implemented in a verification case. Detailed examples of how a unit test can be used to verify models in different contexts are discussed in Collier and Ozik (2013), Asta et al. (2014) and Onggo et al. (2014). Hence, we will not repeat these detailed implementations in this paper. During validation, the behaviour and output of the simulation model can be compared to the expected output. The expected output can be obtained from empirical data or analytic/ theoretical models. In this paper, we validate our ABS model by comparing it with solutions based on analytic models from the search-theory literature. The use of analytic models is useful when empirical validation using complete real data sets is virtually impossible (e.g. we may know the locations and numbers of refugee boats detected annually but not the numbers of undetected refugee boats that manage to land in a protected coastal region).

In this section, we will give four examples of validation cases. The first two cases are based on two hypothetical search strategies commonly used to provide bounds for the performance of a real-world search strategy. The first validation case is used to compare the performance of an *exhaustive search* between the simulation model and the analytical model. In the second case, we repeat the same case but for a *random search*. The remaining cases demonstrate how we validate a real-world search strategy. In the third validation case, we validate a real-world search called a *parallel search* against the lower and upper bounds obtained from analytical models. In the cases mentioned above we assume that *the target is stationary and equally likely to be anywhere in the search area*. This is a

one-sided search which incurs simpler interactions between a searcher and targets. In the fourth case we model the random search of a *mobile target* which seeks to approach the searcher. In this case, the search is two-sided. Both the searcher and targets try to approach each other (more complex interactions than the other three scenarios). We will give more details about each case in subsections 4.1 to 4.4.

To make the explanation easier, all cases in this paper are based on a single-searcher operation and a "definite range" detection model. In the definite-range detection model, detection occurs when the distance between target and searcher is less than the fixed detection range *r*. The total area and search area are squares with sides $\sqrt{A}$ and $\sqrt{A'}$, respectively, and $A' \leq A$. The initial position of a target is assumed to be uniformly distributed over *A*. All simulation results are based on 20 replications.

4.1 Exhaustive search for a stationary target

For a given searcher speed *v*, detection range *r* and sweep width $w = 2r$, with a duration of Δt, a searcher can cover an area of $wv\Delta t$ (we refer to this area as the covered area). An exhaustive search assumes that there is no overlap of one covered area with another, and that no segment of a covered area is placed outside the search area *A'*. The area *A'* will then be completely searched and the target will be detected with probability 1 in time $A'/(vw)$. Hence, an exhaustive search can be thought of as carefully placing a number of non-overlapping covered areas inside *A'*, as shown in Figure 7 (left) (see the confetti analogy in Washburn (2002) p.22). Since the requirement of path continuity in any real-world continuous search case would force gaps and overlaps among covered areas, an exhaustive search should be thought of as an upper bound on the search performance (Washburn 2002).

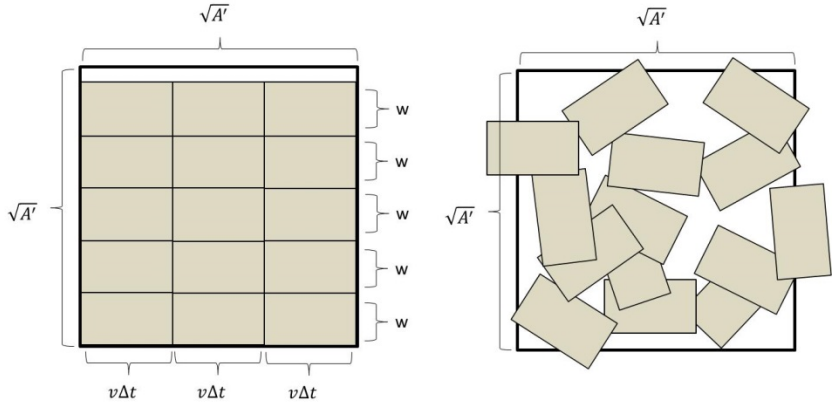


Figure 7: An illustration of Exhaustive search (left); Discrete Random Search (right)

The detection probability of a stationary target depends on the total area covered, which is a linear function of time. Once the search area is completely covered, the detection probability reaches 1 and remains constant. Let *S* be the event that the target is in search area *A'* and *D* be the event that the target is detected. Then, the probability of detecting a stationary target by time *t*, given that the target is located inside *A'* can be written as (Washburn 2002):

$$P_t^{exhaustive}(D|S) = \min\left\{\frac{wvt}{A'}, 1\right\} \quad (1)$$

Figure 8 (the top two curves) shows the performance of an exhaustive search, over time, obtained from the simulation model and the analytic model (equation 1), where $A' = 2{,}304$ nm^2, $v = 20$ kts, $\Delta t = 6$ minutes and $w = 8$ nm. The figure shows a good match between the output of the simulation model and the analytical model. We used equation 1 in our unit test to validate the exhaustive search implementation in our model. For example, with the above settings, we expect that $P(D|S)$ will reach 1 after searching for 864 minutes (i.e. 144 time steps). Our model passes this test, as shown in Figure 6 (*validateExhausticMaxPerformance* case) and Figure 8 (the probability at 864 minutes is close to 1).

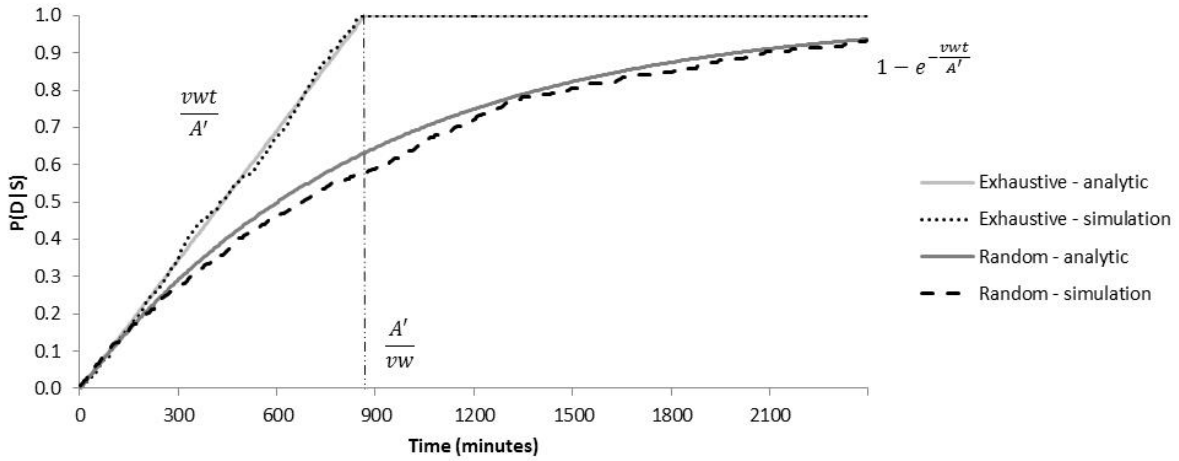
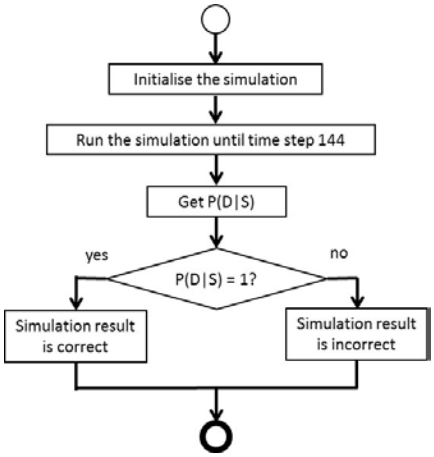


Figure 8: The detection probability ($P(D|S)$) of exhaustive search and discrete random search.

The unit test to represent this validation case is shown in Figure 9. The flowchart is given on the left and the detailed unit test code is given on the right. First, the simulation is initialised (lines 3–5). Next, the simulation is run for 144 steps. This is done by obtaining the schedule from the simulation model which allows us to execute the model in step mode (lines 06–07). This is followed by obtaining $P(D|S)$ from the simulation. To implement this, we introduce agent observer that calculates all system-level statistics. Lines 08-11 find the agent observer from the list of agents in the model. When the agent is found, we can retrieve $P(D|S)$ (lines 12–14). Finally, we test if $P(D|S)$ is close to 1 (line 17). This scenario shows an example of one of the most commonly used validation techniques, in which an analytic result provides an output that is expected from the simulation.




```
01. @Test
02. public void validateExhaustiveMaxPerformance() {
03.   double pds = 0.0;
04.   ResetModel();
05.   builder.SetSearcherMovement(SearcherMovement.EXHAUSTIVE);
06.   ISchedule schedule =
        RunEnvironment.getInstance().getCurrentSchedule();
07.   for (int i=0; i<144; i++) schedule.execute();
08.   ContinuousSpace<Object> space =
        (ContinuousSpace<Object>)context.getProjection("space");
09.   Iterable<Object> listObjects = space.getObjects();
10.   for (Object obj : listObjects) {
11.     if (obj instanceof Observer) {
12.       Observer obs = (Observer) obj;
13.       pds = obs.GetProbDetectedWhenInside();
14.       break;
15.     }
16.   }
17.   assertTrue(Math.abs(pds – 1.0) < 0.0001);
18. }
```

Figure 9: A validation case to check if the output of a simulation model matches the expected value at a specific point in simulation time.

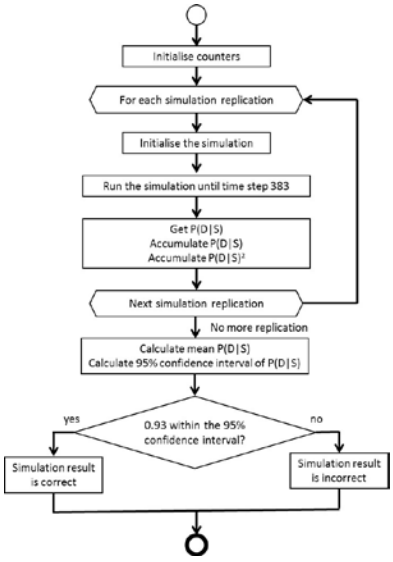
4.2 Random search for a stationary target

Unlike an exhaustive search, a random search places the covered areas randomly inside the search area. The size of a covered area is the same as in an exhaustive search, i.e. $wv\Delta t$. Random placement results in wasted effort because some coverage areas may overlap and some may also cover an area outside the search area (see Figure 7 (right)). Hence, it performs worse than an exhaustive search and provides a lower bound on the performance of any sensible search strategy (Washburn, 2002). The probability of detecting a stationary target by time t, given that the target is present inside the search area, is shown in equation 2. The proof can be found in Koopman (1946).

$$P_t^{random}(D|S) = 1 - e^{-\frac{wvt}{A'}} \quad (2)$$

The performance ($P(D|S)$) of a discrete random search over time is shown in Figure 8 (using the same parameters as in the exhaustive search). The output of the simulation model matches the output of the analytical model. Equation 2 was used in our unit test to validate the implementation of a discrete random search in our model. For example, we can check whether at time 2,298 minutes (i.e. 383 time steps), the analytical result (i.e. 0.93) is within the 95% confidence interval of the simulation result. The simulation model passes the test as shown in the *validateRandomAtTime2298* case in Figure 6.

Figure 10 shows the flowchart and the detailed code that implement this validation case. First, the counters used to calculate the confidence interval are initialised (lines 03–05). Next, we run the simulation a number of times (lines 06–23) so that we can collect the statistics needed for the confidence interval. In each iteration, the simulation is initialised (lines 07–09). Line 08 shows how we can control the random number stream in each simulation run. Next, the simulation is run for 383 steps (lines 10–11). This is followed by obtaining $P(D|S)$ from the simulation (lines 12–17) and collecting statistics that will be needed later (lines 18–19). After we have run all simulation replications, we can form a confidence interval around the mean $P(D|S)$ (lines 24–25). Finally, we test if the analytical result is within the 95% confidence interval (line 26). This scenario shows an example of one of the most commonly used validation techniques, in which a confidence interval is built around a simulation result and we test if the analytic result is within the confidence interval.




```
01. @Test
02. public void validateRandomAtTime2298() {
03.   double sum = 0.0; double sumsq = 0.0;
04.   double pds = 0.0;
05.   double meanPDS, halfCI;
06.   for (int run=0; run<NRuns; run++) {
07.     ResetModel();
08.     builder.SetRandomSeed(run);
09.     builder.SetSearcherMovement(
          SearcherMovement.RANDOM_SEARCH_DISCRETE);
10.     ISchedule schedule =
          RunEnvironment.getInstance().getCurrentSchedule();
11.     for (int i=0; i<383; i++) schedule.execute();
12.     ContinuousSpace<Object> space =
         (ContinuousSpace<Object>)context.getProjection("space");
13.     Iterable<Object> listObjects = space.getObjects();
14.     for (Object obj : listObjects) {
15.       if (obj instanceof Observer) {
16.          Observer obs = (Observer) obj;
17.          pds = obs.GetProbDetectedWhenInside();
18.          sum += pds;
19.          sumsq += pds * pds;
20.          break;
21.       }
22.     }
23.   }
24.   meanPDS = sum/NRuns;
25.   halfCI = 1.96*Math.sqrt(sum_square/NRuns –
        mean_pds*mean_pds)/Math.sqrt(NRuns);
26.   assertTrue(Math.abs(0.93 - mean_pds) < half_ci);
27. }
```

Figure 10: A validation case to check if an analytic result is within the confidence intervals of the simulation result at a specific point in simulation time

4.3 Bounds on parallel search

A parallel search is a real-world continuous search strategy that is frequently employed. The search is conducted by moving along parallel tracks with a separation distance of $G = w$, as shown in Figure 11 (left). For A_S = 2,304 nm^2, v_S = 20 kts, Δt = 6 minutes and w = 8 nm, the performance of a parallel search with $G = w = 8$ is shown in Figure 11 (right). If we stop the simulation at time 864 minutes (i.e. the time when an exhaustive search reaches its maximum performance), we can see that the performance of the parallel search is bounded by the performance of the exhaustive search (upper bound) and the random search (lower bound). Since the performance of any sensible real-world search

is bounded by the exhaustive search and the random search, we can use this in our unit test to validate the performance of any sensible real-world search, as shown in the unit test below (0.632 and 1.0 are the performance of a discrete random search and a parallel search at time 864 minutes or 144 time steps, obtained using analytical models, respectively). Note that the unit test below assumes that the targets are stationary so that we can use the performance bounds given by equations 2 and 3.

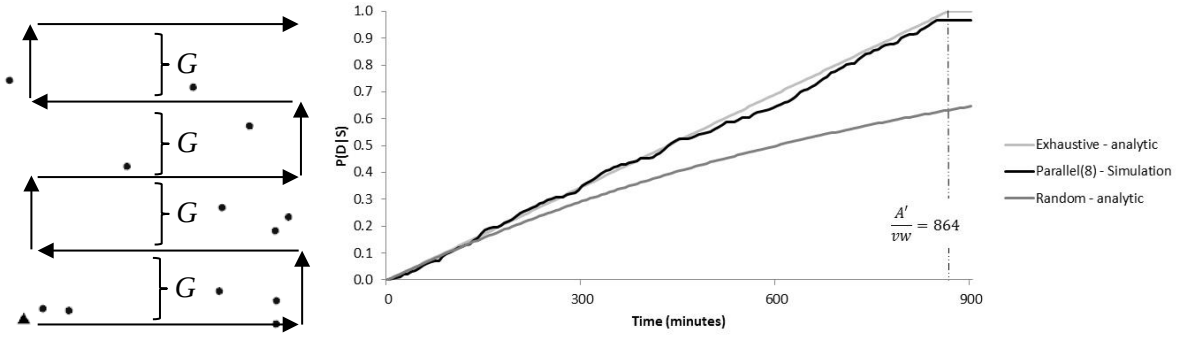
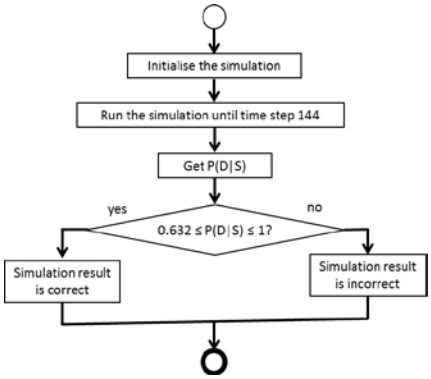


Figure 11: Parallel Search (left: pattern, right: performance with G=w)

The unit test to represent this validation case is shown in Figure 12. It follows a similar algorithm to the one in Figure 9, except for the test condition at the end (line 17). This scenario shows an example of one of the most commonly used validation techniques, in which analytic results provide an upper bound and a lower bound on the simulation result. The model passes this test as shown in the *validateParallelPerformanceBounds* case in Figure 6.




```
01. @Test
02. public void validateExhaustiveParallelPerformance() {
03.   double pds = 0.0;
04.   ResetModel();
05.   builder.SetSearcherMovement(SearcherMovement.PARALLEL);
06.   ISchedule schedule =
        RunEnvironment.getInstance().getCurrentSchedule();
07.   for (int i=0; i<144; i++) schedule.execute();
08.   ContinuousSpace<Object> space =
        (ContinuousSpace<Object>)context.getProjection("space");
09.   Iterable<Object> listObjects = space.getObjects();
10.   for (Object obj : listObjects) {
11.     if (obj instanceof Observer) {
12.       Observer obs = (Observer) obj;
13.        pds = obs.GetProbDetectedWhenInside();
14.       break;
15.     }
16.   }
17.   assertTrue((pds >= 0.632) && (pds <= 1.0));
18. }
```

Figure 12: A validation case to check if analytic results provide upper and lower bounds on the simulation result at a specific point in simulation time

4.4 Random search for a moving target

This scenario represents a more realistic target behaviour. In this scenario, the searcher with detection range r and speed v is searching for a target with counter-detection range k and speed u. We assume that r is less than k since otherwise the target will be detected before he has a chance to react to the searcher. We also assume that u is less than v so that the searcher has a speed advantage. We further assume that the target desires to approach the searcher, i.e. when the searcher falls inside the target's counter-detection range the target moves towards the searcher. This may be because the target might be a victim who hopes to be rescued by the searcher (a search and rescue boat) or an enemy submarine that wishes to attack the searcher (an anti-submarine warfare frigate).

Washburn (2002) provides an equivalent sweep width (w_e), for the searcher in the above scenario, except that the searcher's direction of travel (its course) is fixed throughout the engagement and known to the target. Assuming dimensionless ratios $\alpha = u/v$ and $\beta = r/k$, w_e is derived as follows:

$$w_e = \begin{cases} 2k.\sin(\psi), & \sqrt{\alpha^2 + \beta^2} \leq 1 \\ 2k\,, & \sqrt{\alpha^2 + \beta^2} > 1 \end{cases} \quad (3)$$

where $\psi = \arcsin(\alpha) + \arcsin(\beta)$. Since in our case the target is also moving at speed u, we can extend the result in equation (2) by substituting searcher speed v with the average value of searcher-target relative speed $\bar{v} = \sqrt{v^2 + u^2 - 2uv.\cos\theta}$, where θ is the angle between the velocity vectors (Washburn, 2002). Then the equivalent average speed for the searcher is:

$$\bar{v} \equiv \frac{1}{2\pi}\int_0^{2\pi} \sqrt{v^2 + u^2 - 2uv.\cos\theta}\, d\theta \quad (4)$$

The integral in (4) cannot be evaluated explicitly, therefore it must be computed numerically. The detection probability of a randomly moving target that desires to approach a randomly moving searcher can be computed by substituting w with w_e, and v with the relative speed $\bar{v}$ in equation (2).

$$P_t^{approach}(D|S) = 1 - e^{-\frac{w_e \bar{v} t}{A'}} \quad (5)$$

Note that analytical model in equation (5) is optimistic compared to our simulation scenario. The analytical model assumes that the target can predict the searcher's motion well enough to adopt an intercept, rather than a pursuit, course. Thus the analytical model provides an upper bound for our simulation results. For this validation case, we set A=A' =10,000 nm^2, v = 10 kts., u = 4 kts., r = 3 nm and k = 9 nm. The result is shown in Figure 13 (left). This confirms that the analytical result provides an upper bound to the simulation result. It also shows that the analytical model is significantly more optimistic. Hence, we have also added a validation to the root mean square distance travelled by the searcher in which, analytically, the value is close to $\sqrt{N}$ where N is the number of time steps. Figure 13 (right) shows that the results from the simulation model matches the analytical solution.

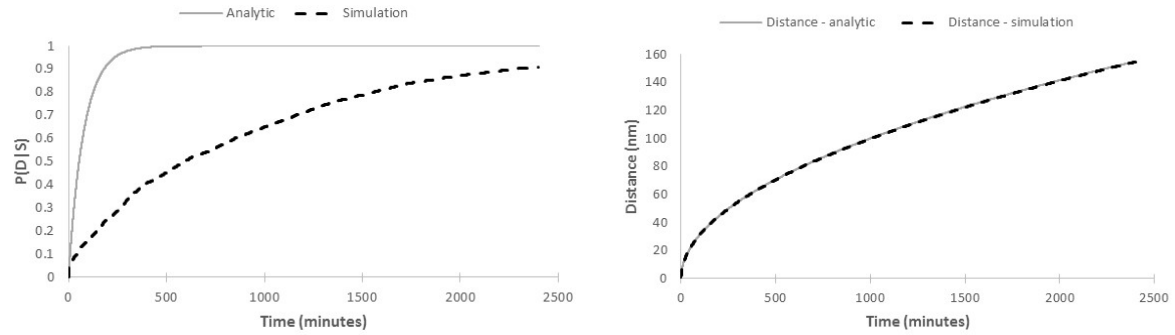
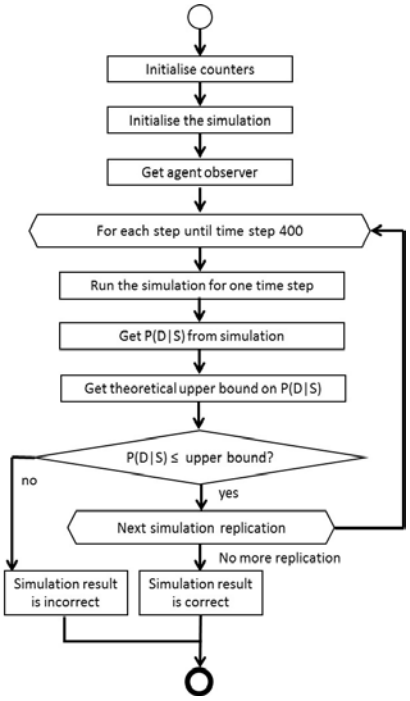


Figure 13: Random search for a moving target (left: detection probability, right: accumulated distance travelled by the searcher)

The flowchart and unit test code that implement this validation case is shown in Figure 14. This scenario shows a different validation technique than in the previous three scenarios in which the simulation result is validated in each time step during a simulation run. This is a tedious process if done manually. Hence, this example underlines one of the advantages of using an automatic validation tool. As in the previous examples, the first steps are the initialisation of the counters and model (lines 03–14). Next, we find object observer to collect $P(D|S)$ from the simulation (lines 15–22). The simulation is run for 400 time steps (lines 23-24). Lines 25 to 32 test whether the simulation

result is less than the upper bound set by the analytical solution. The model passes this test as shown in the *validateRandomMovingTargets* case in Figure 6.




```
01. @Test
02. public void validateRandomMovingTargets() {
03.   double pds = 0.0;
04.   double we = 12.287; v_bar = 10.4;
05.   double analytic_pds;
06.   boolean wrong = false;
07.   ResetModel();
08.   builder.SetSearchArea(10000);
09.   builder.SetSearcherMovement(
      MASSIMConstants.SearcherMovement.RANDOM_SEARCH_CONTINUOUS);
10.   builder.SetSearcherSpeed(10);
11.   builder.SetSearcherDetectionRange(3);
12.   builder.SetTargetMovement(
        MASSIMConstants.TargetMovement.RANDOM_MOVE_APPROACH);
13.   builder.SetTargetSpeed(4);
14.   builder.SetTargetDetectionRange(9);
15.   ContinuousSpace<Object> space =
        (ContinuousSpace<Object>)context.getProjection("space");
16.   Iterable<Object> listObjects = space.getObjects();
17.   for (Object obj : listObjects) {
18.     if (obj instanceof Observer) {
19.       Observer obs = (Observer) obj;
20.       break;
21.     }
22.   }
23.   ISchedule schedule =
        RunEnvironment.getInstance().getCurrentSchedule();
24.   for (int i=100; i<400; i++) {
25.     schedule.execute();
26.     pds = obs.GetProbDetectedWhenInside();
27.     analytic_pds = 1-Math.exp(-we*v_bar*(i+1)/100000);
28.     if (prob_detected_when_inside > analytic_pds) {
29.       wrong = true; break;
30.     }
31.   }
32.   assertTrue(!wrong);
33. }
```

Figure 14: A validation case to check if the simulation result is valid at any point in simulation time

5 CONCLUSION

We have presented a new technique called Test-Driven Simulation Modelling (TDSM). TDSM has a number of benefits. First, it promotes the explicit integration of model verification and validation into the simulation modelling process. Secondly, the cost of finding a mistake early is considerably lower than the cost of detecting, identifying and correcting the mistake later. Finally, for validation, TDSM fits nicely with the black-box validation method in which the output of a simulation model is compared to the output of a benchmark (either from empirical data or an analytical/ theoretical model). The output of an analytical model has been shown to be useful when it is virtually impossible to validate a simulation model against empirical data. We can use analytic models to calculate the performance of simple search strategies. Hence, strictly speaking a simulation model is not needed for the analysis of simple search strategies. However, for a more complex search strategy where an analytic model becomes intractable, we need a simulation model. An analytic model is still useful in the analysis of a more complex simulation model because it can provide us with bounds on performance and helps us with the validation process by making sure that any additional details that increase the complexity of the model do not break existing validation cases. This is shown in the case study using MASSIM in which simpler and more complex search operation cases share the same software components. When a more complex search operation case is added and does not break existing validation cases, our confidence in the correctness of the more complex case increases. There is no guarantee that there will be no mistakes in the more complex case, but at least if there is a mistake, it will be contained within the method/ module that implements the more complex case. Kleijnen (1995) has argued that when a modeller develops a complex simulation model, s/he should validate the model by running a simplified version of the model that has a known analytical solution. In other words, the modeller should be guided by the knowledge of relevant analytic models that are

simpler and have known solutions when they are building a complex model. This paper has shown how this principle is integrated with TDSM.

We have also discussed how ABS can be useful in the analysis of maritime search operations. The application of ABS in maritime search operations is lacking, partly due to the difficulty in validation of an ABS model. Hence, the technique proposed in this paper could help the adoption of ABS as one of the analytical tools in this domain.

There are many challenges in the implementation of TDSM which require further research. First, we need to find out if TDSM could lead to a better model. This is similar to software engineering research that discusses whether TDD could lead to better software, which is inconclusive (Janzen and Saiedian 2005). However, anecdotal evidence based on the incorporation of unit testing in major software development tools (such as JUnit in Eclipse/Java, Visual Studio and NUnit) suggests that TDD may be used by many software developers. Secondly, model verification and validation is a combinatorial problem in which every model requirement often needs a number of verification and validation cases. Hence, it is very expensive to cover all possible mistakes during the model development process. More research is needed to find an optimum strategy for TDSM that balances test coverage and cost. Techniques such as Latin Hypercube described in Thiele et al (2014) can be used to address this problem. Thirdly, a unit test can contain mistakes too. Hence, tools that can help minimise mistakes in unit test are also needed. Finally, the proposed TDSM is appropriate when we can compare the simulation result against the expected behaviour. The expected behaviour can be obtained from empirical data or analytic/ theoretical models. ABS models often deal with heterogeneous agents in which collecting empirical data collection and finding an analytical result that can be used for TDSM are challenging. More research are needed to investigate how TDSM can be useful in this situation. One possible approach is to simplify the models until TDSM can be used. This is consistent with the argument made by Kleijnen (1995) as discussed earlier.

REFERENCES

Akbori, F. 2004 “Autonomous-Agent Based Simulation of Anti-Submarine Warfare Operations with the Goal of Protecting a High Value Unit.” Master’s Thesis, Naval Postgraduate School, Monterey CA.

Asta, S., E. Özcan, and P.-O. Siebers. 2014. “An Investigation on Test Driven Discrete Event Simulation.” In *Proceedings of the Operational Research Society Simulation Workshop*, 35–45.

Balci, O. 1995. “Principles and Techniques of Simulation Validation, Verification, and Testing.” In *Proceedings of the 1995 Winter Simulation Conference*, 147–154.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2010. *Discrete-Event System Simulation*. 5th ed. Upper Saddle River, New Jersey: Pearson Education.

Beck, K. 2003. *Test-Driven Development by Example*. Boston, MA: Addison Wesley.

Bruzzone, A., M. Massei, F. Madeo, F. Tarone, and M. Gunal. 2011. “Simulating Marine Asymmetric Scenarios for Testing Different C2 Maturity Levels.” In *Proceedings of the 16th International Command and Control Research and Technology Symposium*, 12–23.

Champagne, L., R. G. Carl, and R. Hill. 2003. “Agent Models II: Search Theory, Agent-Based Simulation, and U-Boats in the Bay of Biscay.” In *Proceedings of the 2003 Winter Simulation Conference*, 991–998.

Champagne, L.E. 2004. “Development Approaches Coupled with Verification and Validation Methodologies for Agent-Based Mission-Level Analytical Combat Simulations.” Doctoral Dissertation, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, No. AFIT/DS/ENS/03-02.

Cioppa, T.M., T.W. Lucas, and S.M. Sanchez. 2004. “Military Applications of Agent-Based Simulations.” In *Proceedings of the 2004 Winter Simulation Conference*, 171–180.

Coggins, P.B. 1971. “Detection Probability Computations for Random Search of an Expanding Area.” Committee on Undersea Warfare, National Research Council, National Academy of Sciences, Washington, D.C.

Collier, N., and J. Ozik. 2013. “Test-Driven Agent-Based Simulation Development.” In *Proceedings of the 2013 Winter Simulation Conference*, 1551–1559.

Dabrowski, J.J., and J.P. de Villiers. 2015. “Maritime Piracy Situation Modelling with Dynamic Bayesian Networks.” *Information Fusion*, 23: 116–130.

Davidsson, P., L. Henesey, L. Ramstedt, J. Törnquist, and F. Wernstedt. 2005. “An Analysis of Agent-Based Approaches to Transport Logistics.” *Transportation Research Part C: Emerging Technologies*, 13(4): 255–271.

Decraene, J., M. Anderson, and M.Y.H. Low. 2010a. “Maritime Counter-Piracy Study using Agent-Based Simulations.” In *Proceedings of the 2010 Spring Simulation Multiconference*, 165.

Decraene, J., M. Chandramohan, M.Y.H. Low, and C.S. Choo. 2010b. Evolvable Simulations Applied to Automated Red Teaming: A Preliminary Study. In *Proceedings of the 2010 Winter Simulation Conference*, 1444–1455.

DeStefano, G.V. 2004. “Agent-Based Simulation SEAS Evaluation of DoDAF Architecture.” Master’s thesis, Department of the Air Force, Air University, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, No. AFIT/GOR/ENS/04-05.

Duong, D. 2010. “Verification, Validation, and Accreditation (VV&A) of Social Simulations.” In *Proceedings of the 2010 Spring Simulation Interoperability Workshop*, 9 pages.

Gurcan, O., O. Dikenelli, C. Bernon. 2013. “A Generic Testing Framework for Agent-Based Simulation Models.” *Journal of Simulation*, 7: 183–201.

Harney, J.W. 2003. “Analyzing Anti-Terrorist Tactical Effectiveness of Picket Boats for Force Protection of Navy Ships using X3D Graphics and Agent-Based Simulation.” Master’s Thesis, Naval Postgraduate School, Monterey CA.

Harris, S. P., Dixon, D. S., Dunn, D. L., and Romich, A. N. 2013. Simulation Modeling for Maritime Port Security. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 10: 193–201.

Hasegawa, K., K. Hata, M. Shioji, K. Niwa, S. Mori, and H. Fukuda. 2004. “Maritime Traffic Simulation in Congested Waterways and Its Applications.” In *Proceedings of the 4th Conference for New Ship and Marine Technology*, 195–199.

Heath, B., R. Hill, and F. Ciarallo. 2009. “A Survey of Agent-Based Modeling Practices (January 1998 to July 2008).” *Journal of Artificial Societies and Social Simulation*, 12, 9.

Hill, R.R., L.E. Champagne, and J.C. Price. 2004. “Using Agent-Based Simulation and Game Theory to Examine the WWII Bay of Biscay U-boat Campaign.” *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1(2): 99–109.

Hill, R.R., R.G. Carl, and L.E. Champagne. 2006. “Using Agent-Based Simulation to Empirically Examine Search Theory using a Historical Case Study.” *Journal of Simulation*, 1(1): 29–38.

Jakob, M., O. Vaněk, Š. Urban, P. Benda, and M. Pěchouček. 2010. “AgentC: Agent-Based Testbed for Adversarial Modeling and Reasoning in the Maritime Domain.” In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*: Volume 1, 1641–1642.

Jakob, M., O. Vaněk, and M. Pechoucek. 2011. “Using Agents to Improve International Maritime Transport Security.” *Intelligent Systems*, 26(1): 90–96.

Jakob, M., O. Vaněk, O. Hrstka, and M. Pěchouček. 2012. “Agents vs. Pirates: Multi-Agent Simulation and Optimization to Fight Maritime Piracy.” In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* – Volume 1, 37–44.

Janzen D., and H. Saiedian. 2005. “Test-Driven Development: Concepts, Taxonomy, and Future Direction.” Computer, 38(9): 43–50.

Karataş, M. 2012. “An Analytical Comparison of Random and Exhaustive Search of an Expanding Area with Binary Sensors.” *Engineer & the Machinery Magazine*, 23(4): 2–13.

Kleijnen, J.P.C. “Verification and Validation of Simulation Models.” *European Journal of Operational Research*, 82: 145–162.

Klugl, F. 2008, “A Validation Methodology for Agent-Based Simulations.” In *Proceedings of the 23rd Annual ACM Symposium on Applied Computing*, 39–43.

Koopman, B.O. 1946. Search and Screening. OEG Report No. 56.

Law, A.M. 2014. *Simulation Modeling and Analysis*. 5th ed. Boston, MA: McGraw-Hill.

Leathrum, J.F., R. Mathew, and T.W. Mastaglio. 2009. “Modeling and Simulation Techniques for Maritime Security.” In *Proceedings of the Conference on Technologies for Homeland Security*, 636–642.

Macal, C.M., and M.J. North. 2010. “Tutorial on Agent-Based Modelling and Simulation.” *Journal of Simulation*, 4: 151–162.

Marchione, E., S.D. Johnson, and A. Wilson. 2014. “Modelling Maritime Piracy: A Spatial Approach.” *Journal of Artificial Societies and Social Simulation*, 17(2): 9.

McCard, A. n.d. “Port of Jacksonville uses Simulation for Terror Threat Training.” [online] Available at: http://www.ems1.com/ems-products/educationtraining/simulation/press-releases/233816-Port-of-Jacksonville-Uses-Simulation-For-Terror-Threat-Training/ (accessed 30 June 2015).

Metron Incorporated. 2015. “Naval Simulation System (NSS).” [online] Available at: http://www.metsci.com/Division/ORCA/Naval-Simulation-System-NSS-2 (Accessed 16 June 2015).

Ng, C.W. 2007. “Discrete-Event Simulation with Agents for Modeling of Dynamic Asymmetric Threats in Maritime Security.” Master’s thesis, Naval Postgraduate School, Monterey, CA.

Niazzi M.A., A. Hussain, and M. Kolberg. 2009. “Verification & Validation of Agent-Based Simulations using the VOMAS (Virtual Overlay Multi-Agent System) Approach.” In *Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops*, 494–501.

North, M.J., and C.M. Macal. 2007. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. New York: Oxford University Press.

North, M.J., N.T. Collier, J. Ozik, E.R. Tatara, C.M. Macal, M. Bragen, and P. Sydelko. 2013. “Complex adaptive systems modeling with Repast Simphony”. *Complex Adaptive Systems Modeling*, 1(1): 3.

Nunn, L.H. 1981. “An Introduction to the Literature of Search Theory.” Technical Report, Center for Naval Analyses Alexandria VA, Operations Evaluation Group, No. CNA-PP-305.

Onggo, B.S.S., C. Indriany, and M. Gunal. 2014. “Test-Driven Simulation Modelling.” In *Proceedings of the 7th International Workshop on Applied Modeling and Simulation (WAMS14)*, Istanbul, Turkey, edited by Bruzzone, Çayırcı, Günal, Kaylan, Massei, Zanni-Merk. 43–48.

Ormerod P., and B. Rosewell. 2006. “Validation and Verification of Agent-Based Models in the Social Sciences.” In *Epistemological Aspects of Computer Simulation in the Social Sciences*, Lecture Notes in Computer Science Volume 5466, 130–140. Berlin, Heidelberg: Springer-Verlag.

Pidd, M. 2004. *Computer simulation in management science*. 5th ed. Chichester, England: John Wiley & Sons.

Price, J.C. 2003. “Game Theory and U-boats in the Bay of Biscay.” Master’s thesis, Department of the Air Force, Air University, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, No. AFIT/GOR/ENS/03-18.

Sargent, R.G. 2013. “Verification and Validation of Simulation Models.” *Journal of Simulation*, 7: 12–24.

Shieh, E., B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. 2012. “PROTECT: A Deployed Game Theoretic System to Protect The Ports of the United States.” In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, Volume 1, 13–20.

Steele, M.J. 2004. “Agent-Based Simulation of Unmanned Surface Vehicles: A Force in the Fleet.” Master’s Thesis, Naval Postgraduate School, Monterey, CA.

Sullivan, P.J. 2006. “Evaluating the Effectiveness of Waterside Security Alternatives for Force Protection of Navy Ships and Installations using X3D Graphics and Agent-Based Simulation.” Doctoral dissertation, Naval Postgraduate School, Monterey, CA.

Thiele, J.C., Kurth, W., and Grimm V. 2014. “Facilitating Parameter Estimation and Sensitivity Analysis of Agent-Based Models: A Cookbook Using NetLogo and R.” *Journal of Artificial Societies and Social Simulation*, 17(3): 11.

Tsilis, T. 2011. “Counter-Piracy Escort Operations in the Gulf of Aden.” Master’s thesis, Naval Postgraduate School, Monterey, CA.

Xing, D., H. Wan, M.Y. Zhu, S.M. Sanchez, and T. Kaymal. 2013. “Simulation Screening Experiments using Lasso-Optimal Supersaturated Design and Analysis: A Maritime Operations Application.” *In Proceedings of the 2013 Winter Simulation Conference*, 497–508.

Vaněk, O., B. Bošanský, M. Jakob, and M. Pěchouček. 2010. “Transiting Areas Patrolled by a Mobile Adversary.” In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG)*, 9–16.

Vaněk, O., M. Jakob, O. Hrstka, and M. Pěchouček. 2012. “Using Multi-Agent Simulation to Improve the Security of Maritime Transit.” In *Multi-Agent-Based Simulation* XII, 44–58. Berlin, Heidelberg: Springer-Verlag.

Vaněk, O., M. Jakob, O. Hrstka, and M. Pěchouček. 2013. “Agent-Based Model of Maritime Traffic In Piracy-Affected Waters.” *Transportation Research Part C: Emerging Technologies*, 36: 157–176.

Varol, A.E., and M.M Gunal. 2015. “Simulating Prevention Operations at Sea against Maritime Piracy.” *Journal of the Operational Research Society*. doi:10.1057/jors.2015.34

Walton, D. J., E.P. Paulo, C.J. McCarthy, and R. Vaidyanathan. 2005. “Modeling Force Response to Small Boat Attack against High Value Commercial Ships.” In *Proceedings of the 2005 Winter Simulation Conference*, 988–991.

Washburn, A.R. 1980. “Expanding Area Search Experiments.” Technical Report, Naval Postgraduate School, Monterey, CA, No. NPS55-80-017

Washburn, A., and R. Hohzaki. 2001. “The Diesel Submarine Flaming Datum Problem.” *Military Operations Research*, 6(4): 19–30.

Washburn, A. R. 2002. *Search and Detection*. 4th edition. Linthicum, Maryland, USA: Institute for Operations Research and the Management Sciences.

Windrum, P., G. Fagiolo, and A. Moneta. 2007. “Empirical Validation of Agent-Based Models: Alternatives and Prospects,” *Journal of Artificial Societies and Social Simulation*, 10(2): 8 pages.