A Multiobjective Optimisation Approach For The Dynamic Inference and Refinement Of Agent-Based Model Specifications

Salem F. Adra
Microsoft
STC
London, UK
sadra@microsoft.com

Mariam Kiran
School of Computing
University of Leeds
Leeds, UK
m.kiran@leeds.ac.uk

Phil McMinn
Dept. of Computer Science
University of Sheffield
Sheffield, UK
p.mcminn@sheffield.ac.uk

Neil Walkinshaw
Dept. of Computer Science
University of Leicester
Leicester, UK
n.walkinshaw@mcs.le.ac.uk

***Abstract*—Despite their increasing popularity, agent-based models are hard to test, and so far no established testing technique has been devised for this kind of software applications. Reverse engineering an agent-based model specification from model simulations can help establish a confidence level about the implemented model and in some cases reveal discrepancies between observed and normal or expected behaviour. In this study, a multiobjective optimisation technique based on a simple random search algorithm is deployed to dynamically infer and refine the specification of three agent-based models from their simulations. The multiobjective optimisation technique also incorporates a dynamic invariant detection technique which serves to guide the search towards uncovering new model behaviour that better captures the model specification. The Non-dominated Sorting Genetic Algorithm (NSGA-II) was also deployed to replace the random search algorithm, and the results from both approaches were compared. While both algorithms revealed good potential in capturing the model specifications, the pure exploratory nature of random search was found more suitable for the application at hand, compared to the balanced exploitation/exploration nature of genetic algorithms in general.**

I. Introduction

Computational models are computer programs designed to simulate complex systems; for example financial markets and natural systems such as skin tissue and insect colonies. Scientists and industrialists use computational models to help develop their understanding of the natural system being modeled, to make forecasts, and to predict the impact of some changes to the system. With this in mind, it is of high importance that the models have been properly tested. Recent scientific software errors have led to papers being retracted from *Science* [2]. Empirical work by Hatton [3] found an average of eight serious faults in every 1000 lines of C code analyzed in a series of large scientific programs. In the banking sector, losses made by NatWest, Barclays and Deutsche Morgan Grenfell totalling tens of millions of pounds were blamed on decisions that involved economic model errors [4].

Agent-based modelling is attractive because it allows a complex system to be constructed in terms of the rules that govern individuals within that system. The macroscopic behaviour "emerges" from low-level interactions between the individual agents. Agent-based models (ABMs) have been used to model a diverse range of complex systems, from skin-cell colonies [9] to the behaviour of humans in a crowd, up to full-scale models of macro-economic systems [5].

Emergence - the very phenomenon that makes agent-based models so powerful - also makes them very difficult to test. Emergent behaviour, though intended, is not explicitly specified; it arises out of complex (and often unexpected) interactions between individual agents. How do we know that the individual agents are behaving as they are supposed to? What inputs (i.e. initial agent-states) would it take to cause the system to misbehave - to produce emergent behaviour that is unexpected or incorrect?

The predominant current approach to answering these questions is to test the system by hand, on an ad-hoc basis. The developer manually selects some initial configurations for the system, and observes the ensuing system behaviour via visualisations. This is both time-consuming, and virtually futile if the system contains lots of agents. There may be thousands of agents, each of which has a virtually unlimited number of initial configurations, that can collectively yield a vast range of possible outcomes.

This paper presents a dynamic framework, to intelligently probe these large systems and reverse engineer their specifications. This work builds on the previous work presented in [6] by making the inference process dynamic, which allows the refinement of the inferred specifications. The idea is to explore the input space of ABMs in search for model configurations that reveal new behaviour (i.e. outputs). The framework uses an inference technique called Daikon to observe and learn simple rules about the general model behaviour. It combines this with a multi-objective random search algorithm to seek out configurations that will break the inferred rules. The main objective of this process is to converge or get as close as possible to the real underlying specification of a certain ABM. Such inferred specification can then be reported to the model designer to assist him/her validating the functionality of the model.

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

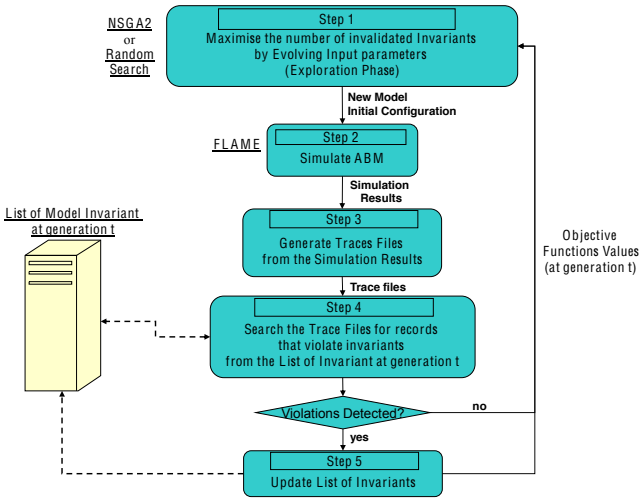

Fig. 1. The Dynamic Framework for Reverse Engineering ABM Specifications

This paper is structured as follows: In Section 2, the proposed framework for reverse engineering ABM specifications is introduced in detail. In Section 3, the experimental setup used to evaluate the performance of the suggested framework is discussed. Section 4 then presents experimental results with the framework for three models. Section 5 presents related work, while Section 6 conclusions and discusses directions for future work.

II. The Proposed Framework

In Figure 1, the implemented framework for dynamically inferring and refining ABM specifications from model simulations is presented. The framework deploys two essential processes: 1) a multiobjective metaheuristic search process, and 2) a dynamic invariant detection process used to evaluate and score the different objective functions. Model, or more generally, program invariants are essential information that can be found in software documentation, or even in comments or assert statements in the code itself. Such model invariants are crucial for defining the model specification, but are nevertheless often missing or unknown. Reverse engineering tools such as the dynamic invariant detector *Daikon* [7] play an important role in proposing any such invariants that can further help the software programmer or designer gain more insights on the software functionality.

In Figure 1, Step 1 is the exploration phase which operates in the decision variable space (i.e. input space) of an ABM and which is essentially implemented as a random search (NSGA2 [1] is also investigated - see Section Experimental Setup). At step 1, a new population of decision vectors that can be used to configure different instances of the ABMs investigated are produced. The objective of Step 1 is to search for interesting ABM inputs (i.e. initial agent state configurations) that might lead the simulated model to violate invariants from the archived list of invariants. Steps 2-5 can be seen as the processes of calculating the objective functions and updating the archive of best (i.e. most general and refined) model invariants found so far. The archive of best solutions found (i.e. the list of model invariants in Figure 1) is initially empty, and the model invariants found at the first generation of the process are first copied to this archive (Step 5). Subsequently, this archive is dynamically updated to only include the most general invariants that subsumes previously inferred invariants, and which more closely capture the model specifications being inferred. More closely, in Step 2 (the 1st stage of the objective function evaluation process), several instances of the investigated ABM are simulated for a certain number of iterations using each of the candidate vectors of model configurations produced in Step 1. In this work, the agent-based framework used to simulate the agent-based models investigated is the Flexible Large-scale Agent-based Modelling Environment (FLAME) [8]. Using FLAME, the model simulation outputs are stored in XML files at each iteration of the model simulation. At Step 3 (the 2nd stage of the objective function evaluation process), a trace file is compiled from the execution of the different ABM instances at Step 2. These trace files contains the concatenated data stored in the produced XML files for each model execution. The dynamic invariant detector Daikon [7] is then used to read and process the trace files at Steps 4 and 5. Daikon is a popular and widely used tool for dynamic detection of likely program invariants. At the first generation of the framework, step 4 is skipped, since the archive is initially empty, and step 5 is executed on the trace files to create an initial list of invariants to populate the archive. These invariants are observations about some traced agent variables and which held true during an execution (e.g. $x > 5$ or $0 < y < 100$). Subsequently at the next generations of the framework, step 4 is executed by calling Daikon to check whether any trace file contain data that invalidate some invariants stored in the archive. The different model configurations produced at Step 1 are then ranked, with the best ranks being allocated to the candidate decision vectors of model configurations violating the most invariants. The whole process is then repeated for a certain number of generations.

A multiobjective approach was adopted for the exploration search process at Step 1 in order to cooperatively and efficiently search for inputs that simultaneously optimize multiple objective functions. For each type of agent involved, an objective is created where the goal is to falsify that agent's inferred invariants, therefore dynamically refining the automatically inferred model specifications.

III. Experimental Setup

In this section, the experimental setup used to evaluate the framework described in the previous section is discussed.

A. The Investigated ABM Model

Three agent-based models were used to assess the performance and utility of the proposed framework: a simple Fox-Rabbit (also widely known as the predator-prey) model, a biological agent-based model which simulates the formation

TABLE I
FOX RABBIT MODEL: AGENTS' MEMORY VARIABLES

x (m)	x coordinate
y (m)	y coordinate
lifeExpectancy	life expectancy of a Fox Agent
numberOfEatenPreys	Number of rabbits eaten by a Fox Agent

TABLE II
KERATINOCYTE MODEL: AGENTS' MEMORY VARIABLES

x (m)	x coordinate
y (m)	y coordinate
type (m)	Type
cycle (m)	Position in Cell Cycle
z	z coordinate
force_x	Forces exerted by other surrounding agents on an Agent in the x direction
force_y	Forces exerted by other surrounding agents on an Agent in the y direction
force_z	Forces exerted by other surrounding agents on an Agent in the z direction
num_xy_bonds	Number of lateral bonds
num_z_bonds	Number of vertical bonds
num_stem_bonds	Number of bonds with Stem Cells
contact_inhibited_ticks	Number of iterations during which an Agent was contact inhibited
diff_noise_factor	Differentiation noise factor
motility	Agent motility
dir	Agent migration direction

of skin tissue [9] (also referred to as the keratinocyte or skin model from this point onward) and an economic model. The Fox-Rabbit model involves fox (predator) and rabbit (prey) agents, the number and ratios of which can vary. In this study, the number of rabbits and foxes were fixed to 100 and 20 respectively. This was done in order to have a better idea of how such model is expected to behave under fixed conditions. The functionality of the fox and rabbit agents is quite simple. Fox agents chase and eat rabbits, while rabbits try to dodge and run away from foxes.

TABLE III
ECONOMIC MODEL: AGENTS' MEMORY VARIABLES

Firm Agent	
x (m)	x coordinate
y (m)	y coordinate
Productivity (m)	Rate of production
Production (m)	Firm's Production
Profits	Firm's Profit
Person Agent	
Savings (m)	Person's Savings
Wage (m)	Person's Wage
x (m)	Person's x location
y (m)	Person's y location
Mall Agent	
x (m)	x coordinate
y (m)	y coordinate

The biological agent-based model is on the other hand more complex. It involves 3 main types of interacting cell agents (stem, transit amplifying (TA) and committed cells (COM)). Each type of cell agent has a specific functionality, and their collaboration and interaction lead to the formation of virtual piece of skin tissue. For more information about the keratinocyte colony formation model, the interested reader is directed to [9]. For this study, the number of initial, randomly generated and seeded cell agents was set to 5. Depending on their locations and types, these cells can then divide, migrate or die during model simulations.

The third model investigated is an economic agent-based model that allows interactions between 3 types of agents: malls, firms and persons. The simulation starts by the Firms and Persons becoming aware of their surrounding malls. Once they have this detail the person agent send job applications to the malls and the firms send vacancy messages to the malls. The mall agents are responsible for getting this information and sorting which person works at which firm based on their wage demand and distance (location). In the second phase of this model the malls also collect products from the firms to sell them to the person agents. The malls hence buy the goods from the firms and sell them to person agents at the highest possible price in the market. Therefore the economic model simulates an active labour market where people are employed to work at firms and get paid wages. The model also simulates a goods market where the firms sells their products to the malls who in turn sell it to the people on the system.

In Tables I, II and III, the different agents' memory variables are illustrated for all three models respectively. All these variables were traced, and the variables postfixed with (m) were also manipulated by the random search or NSGA2 at Step 1.

B. Framework Setup

In addition to using a random search to explore the ABM input domains, NSGA2 was also used in a second set of experiments in order to compare the results produced by the two stochastic algorithms. Because of the iterative and population based nature of the deployed random search and NSGA2, and due to the fact that the application investigated involved simulating computationally expensive ABMs to calculate objective functions, the population sizes and number of generations for each algorithm were set to 10 and 20 respectively. It should be noted, that this setting is much lower than the usual settings used with NSGA2 (usually population sizes of 100 and 250 generations), but for this proof of principle and experimental paper these settings were deemed a good start. For NSGA2, the crossover operator used in step 1 consisted of a simple two-parent, single point crossover which produced two new offspring solutions by swapping values of their respective parent solutions. Moreover, the mutation operator used in NSGA2 operated on individual candidate solutions by randomly mutating their manipulated variables (see Tables I, II and III) within their domain of definition. In Table IV, the configuration parameters used for the Steps 1 to

4 are presented. Except for the use of crossover and mutation operators, the same framework configuration presented in Table IV was used for the random search.

C. Framework Evaluation

In order to assess the performance of the suggested framework, 30 executions of the random search (RS) and NSGA2 based frameworks were executed using the configuration presented in Table IV. The final list of invariants produced by each execution *i* of RS is then assessed with respect to the results produced by NSGA2. This is performed by searching all of the trace files produced by the 30 executions of NSGA2, for records (agents' data) that violated the invariant list produced by a certain execution *i* of RS. The total number of all such violations is then used to score the list of invariants produced by RS at execution *i*. The same process is then reversed to assess the quality of the final list of invariants produced by each execution of NSGA2. The invariant lists produced by a certain algorithm and that were less violated by the other algorithm were then deemed better, and hence more general and closer to capturing the specification of a certain ABM. Moreover, the same comparative result assessment process was conducted on a generational basis to monitor how both algorithms' search progressed towards a certain final list of invariants. This was performed by assessing the list of invariants produced by a certain algorithm at generation *j* (j= 1 to 20) of a certain execution *i* (i = 1 to 30) by counting the number of records violating it and which originate from the trace files produced at generation *j* of all executions *i* of the other algorithm. In the next section, the results produced by each algorithm for each ABM are presented and discussed.

IV. Results

In Figures 2, 3 and 4 box plots are used to statistically depict the quality of the final results produced by each algorithm with respect to the other for each of the 3 agent based models. In Figure 2, the lower, median and upper quartiles illustrating the number of RS Traces violating NSGA2 and vice versa for the Fox (predator) and Rabbit (prey) agents are illustrated. For the Fox agents, RS produced a median value of approximately 800 violations of the final invariants lists produced by NSGA2 for Fox agents, compared to a median value of around 400 violations for NSGA2, i.e. half the number of violations produced by RS. The same observation was illustrated for the Rabbit agents. RS was producing almost double the number of violations compared to NSGA2 with respect to the invariant lists produced at each of the 30 executions for the rabbit agents. This was clearly indicating that RS was producing better list of invariants for the rabbit and fox agents, and which can be used by the model designer to verify the functionality of the model and check for any inconsistencies or clear anomalies. Another remark which is worth noting as well, is that the size of the box plots produced by RS for the fox-rabbit agent model were clearly larger than the box plots produced by NSGA2 which was expected due to the random search nature of RS compared to

TABLE IV
Framework Configuration

NSGA2 Operators	
Crossover Probability	**0.8**
Mutation Probability	**0.2**
Random Search and NSGA2	
Population size	**10**
Number of generations	**20**
Number of Model iterations (Step 2) for each of the 10 candidate solutions	**100**
Fox Rabbit Model	
Objective Functions	
1	**Maximize the number of Fox agents' invariant violations**
2	**Maximize the number of Rabbit agents' invariant violations**
Variables Range	
x **y**	**[10, 990]** **[10, 490]**
Keratinocyte Model	
Objective Functions	
1	**Maximize the number of STEM agents' invariant violations**
2	**Maximize the number of TA agents' invariant violations**
3	**Maximize the number of COM agents' invariant violations**
Variables Range	
x **y**	**[10, 990]** **[10, 490]**
type	{**0, 1, 2**} 0 = Stem 1 = TA 2 = COM
cycle	**[0, 120] for Stem agents** **[0, 60] for TA agents** **0 for COM agents**
Economic Model	
Objective Functions	
1	**Maximize the number of Firm agents' invariant violations**
2	**Maximize the number of Person agents' invariant violations**
3	**Maximize the number of Mall agents' invariant violations**
Variables Range	
x **y**	**[10, 990]** **[10, 490]**
Productivity	**[1, 50]**
Production	**[1, 50]**
Wage	**[0, 50]**
Savings	**[0, 10]**

the more structured and directed search process that NSGA2 was performing. Nevertheless, and within the computational resources afforded to this exploratory study, the exploration process of RS was clearly more convenient to the application at hand, compared to the balanced exploration/exploitation approach of evolutionary algorithms in general.

In Figures 3 and 4, the same observations presented for the fox-rabbit model were observed for the skin and economic agent based models. In fact, because the economic and skin models involved 3 objective functions (i.e. 3 agents' invariant lists to build) compared to the two objective nature of the fox-rabbit model, and because of the increased complexity of the economic and skin models compared to the fox-rabbit model, RS seemed to even better perform compared to NSGA2 for these last 2 models. Exploration, even a random one, was clearly better valued for capturing the behaviour of these models.

In Figures 5, 6 and 7, the generational behaviour of RS and NSGA2 is illustrated for each of the 3 agent based models. The idea was to compare the quality of the list of invariants compiled at each generation of the algorithms with respect to the quality of the list of invariants produced by the other algorithm at the same generation of each of the 30 runs. In Figures 5, 6 and 7, the average (out of 30 runs) cumulative number of violations produced by a certain algorithm with respect to the other in terms of a certain type of agents was presented the 20^{th} generation of each run.

For the relatively simple fox-rabbit model, it was observed that after the 6th generation (out of a total of 20) of each of the 30 runs, no new invariant violation were produced by both algorithms. In other words, the final list of invariants produced after 20 generations for the fox-rabbit model by RS and NSGA2 was the same list of invariants already compiled at the 6th generation. From Figure 5, it can be seen that RS and NSGA2 were both competitive, especially the first 2 to 4 generations, at exploring new model invariants, but RS due to its pure random exploration nature was doing better the last few generations (3 to 6).

In Figure 6, for the more complicated skin model, NSGA2 was doing better than RS at the beginning of the search process (the first few generations) and was producing higher quality agent invariants for the 3 types of agents, especially the commited cell agent type (COM). Nevertheless, RS was eventually producing more general invariants towards the end of the search process (i.e. towards the last generations). This was again highlighting that for the computationally expensive nature of the objective functions used in this study, RS was a better choice, although evolutionary algorithms are expected to do better if more number of generations and/or an increased population of candidate solutions were provided. This would however be practical if metamodeling techniques are deployed to substitute the expensive objective functions used and which consist of simulating agent-based models for a number of iterations, or if higher computing resources such as grid or supercomputing are afforded. These suggestions will be investigated in future work.

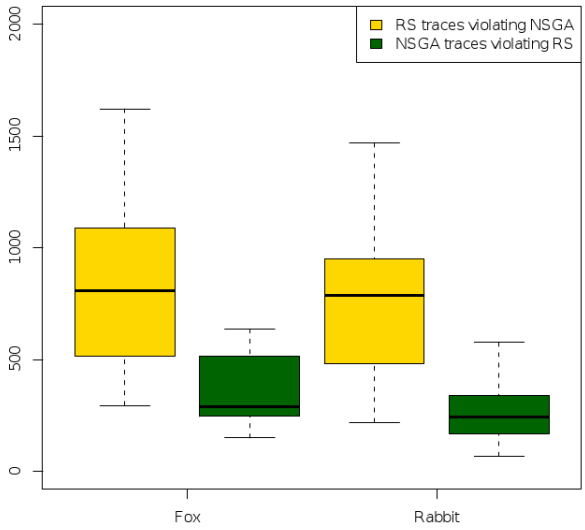

Fig. 2. No of invariant violations: Fox-Rabbit Model

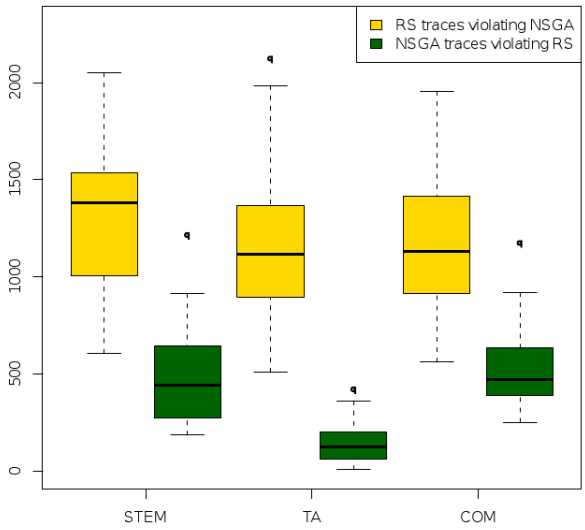

Fig. 3. No of invariant violations: Skin Model

In Figure 7, similar observations to the one depicted in Figures 5 and 6 are shown for the economic model. In fact, for this ABM, RS seemed to be even more consistent and suitable for exploring the input domain of the economic model, which ultimately led to producing more invariant violations compared to NSGA2.

V. Related Work

In [10], the authors used search based testing to test the commercial FIFA95 computer game which involves a set of human controlled agents competing against a set of computer controlled agents. In [10], the objective was to exercise the game in order to find possible sequences of player inputs (controls) that can undermine the enjoyability of the game by

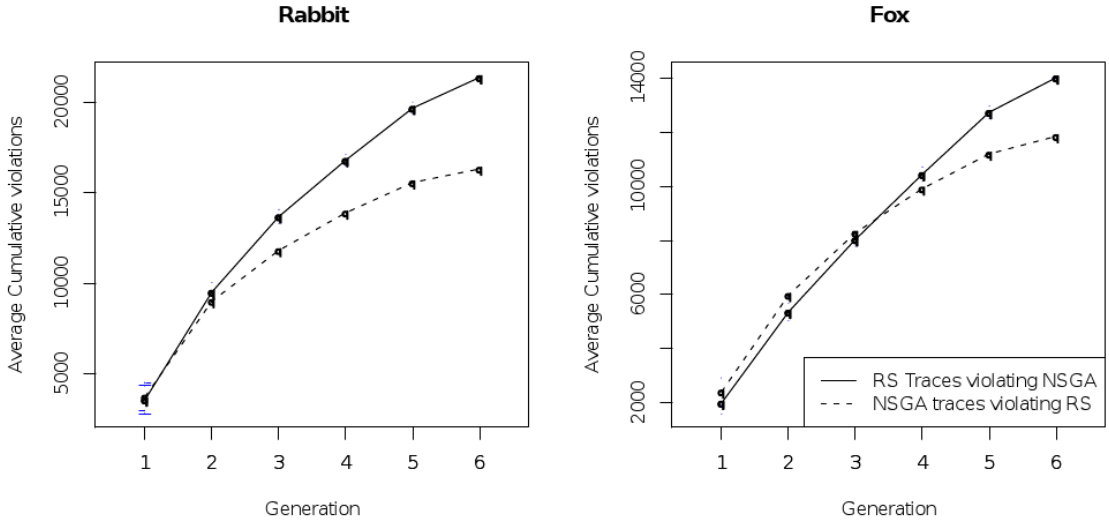

Fig. 5. Average Cumulative No of invariant violations per generation: Fox-Rabbit Model

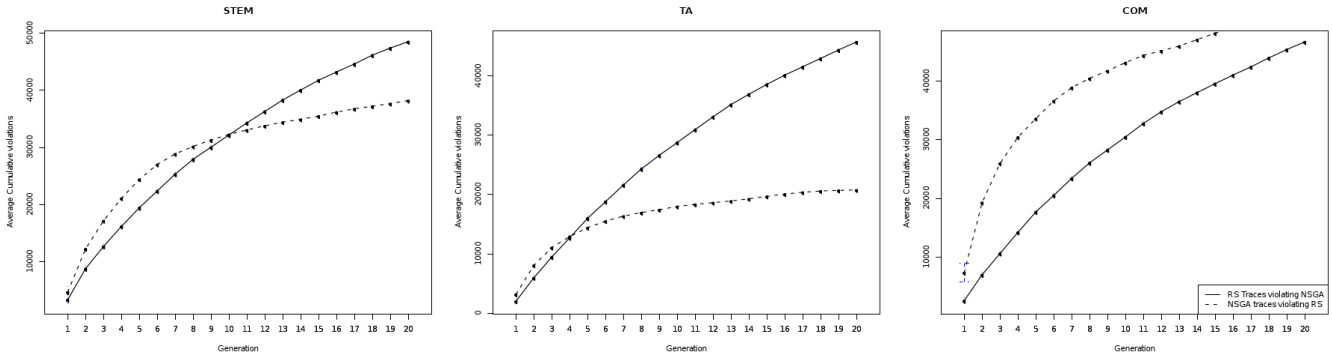

Fig. 6. Average Cumulative No of invariant violations: Skin Model

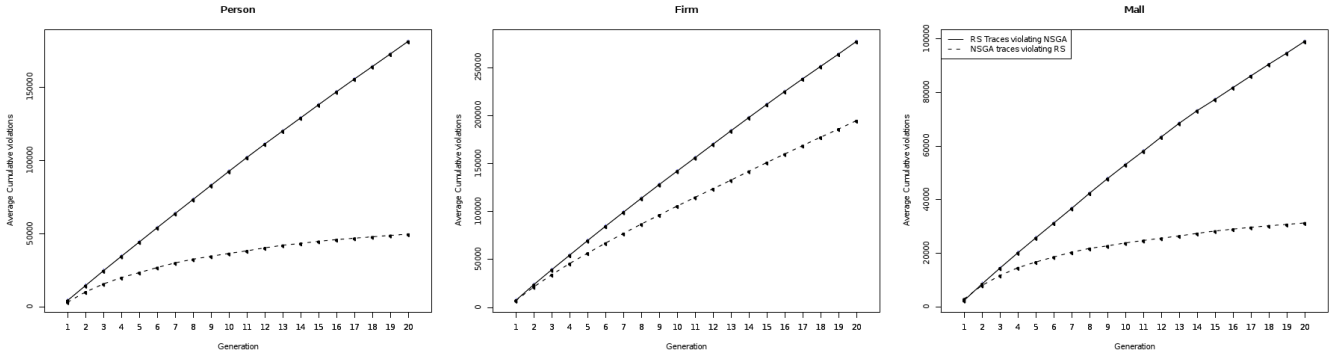

Fig. 7. Average Cumulative No of invariant violations: Economic Model

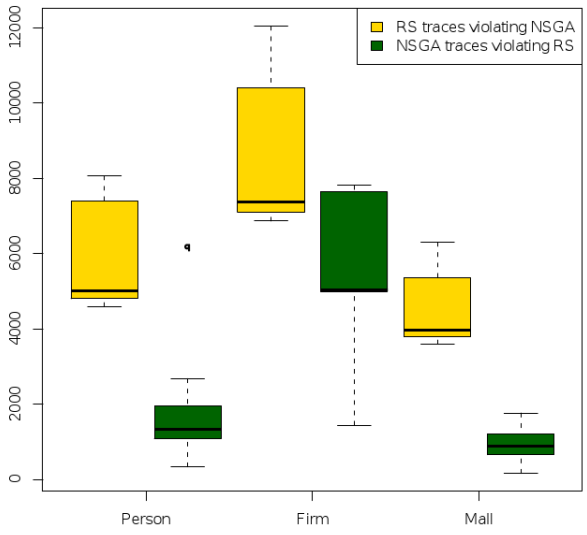

Fig. 4. No of invariant violations: Economic Model

repeatedly allowing to score an easy goal. The authors used a well-tuned genetic algorithm to search for such sequences and were successful in identifying sweet spots of action sequences that consistently led to scoring a goal. Similarly, in [12], [11], the authors tested an ABM which simulated a city struck by an earthquake and where autonomous software agents are taught to evade danger, rescue people and maintain a certain energy level to survive. The authors tested the design of these autonomous agents by injecting into their virtual worlds mock agents that were cooperative and helpful at some times, and aggressive at other times. The action sequences of the mock agents were evolved using a genetic algorithm, and the study succeeded in uncovering some unwanted agent behavior and design weaknesses.

In [13], the authors proposed the recruitment metaphor to evaluate "goal-oriented" autonomous agents in particular. In other words, agents were seen as job candidates and stakeholder requirements (i.e soft goals such as efficiency, accuracy, reliability, ...) were used as evaluation criteria and to formulate fitness functions. Their testing objective was then to use SBST to search for increasingly demanding test cases which can reveal how useful a certain agent is.

In a related study by Bongard *et al.* [14], the authors used a coevolutionary algorithm in a non-linear system identification application. More closely, the authors co-evolved a population of models that approximate a targeted system, and a population of tests that exercise these candidate models. The tests were aimed at extracting new outputs from the candidate models (which mismatch the output responses of the target system to these same tests). Their objective was to refine the candidate models to better approximate the targeted system, as opposed to testing agent-based models.

In [15] and [16], the authors introduced techniques for the verification of Agent-Based models. In [15], a semi automatic methodology was suggested for verifying Agent-based simulations and it involved human expert interaction for model output validations. In [16], in order to verify and validate the simulations of ABM, the authors suggested the use of an overlay layer of listener agents that can be injected into the simulation to collect and log agents' data and report the violations of certain predefined conditions.

Other suggested techniques involving unit testing were also proposed to test and validate ABMs. These include the use of unit tests [17] and the use of mock agents as unit tests [18] for testing specific agents' functionality. In [19], the authors suggested the use of mutation testing as a way for evaluating the reliability of agent-based models, and suggested prototypes for ABM customized mutation operators.

VI. Conclusions and Future Work

In this empirical study, a multiobjective stochastic search approach for inferring and refining Agent based models' specifications is suggested and implemented. The approach is specifically useful for establishing confidence in agent based model simulations/predictions in situations where the modelers' knowledge and feedback about the expected behavior of a certain model is rather limited. The proposed framework can also serve as an automatic feedback mechanism which can help the model's designer to gain more insight on the model internal structure and functionality. Such information might then be used by the modelers to further refine the model or even steer new experimental work. Random Search and Evolutionary algorithms were deployed as the core search process in this framework, and given the computationally expensive nature of using model simulations as objective functions, in this study, we had to limit the number of generations and size of the manipulated populations of candidate solutions to 20 and 10 respectively. These configurations are rather limited for an evolutionary search process, and as a result, the pure random search process was found to better perform and produce higher quality list of model invariants which more closely capture the behaviour of a model. The framework was however proved useful at automating the process of reverse engineering and refining model specification from model simulations, and thus showed a lot of potential for being refined and used by system modelers for establishing confidence in a certain model prediction, a requirement very much important and beneficial. This method of verification can be compared to previous research where unit testing [18] and [15] use a human interaction to test the results. By automating the process using the framework, invariants and rules can be identified easily to deduce if the model has performed as not expected by the modeler.

As a future work, there is scope to investigate assigning different weights and priorities among the different objectives being optimised (i.e. the different agents' whose invariants are sought to to captured and refined). In addition, meta-modeling techniques will be investigated to substitute the

use of simulating agent based models as objective functions. This will make it more convenient and practical to deploy intelligent search algorithms such as evolutionary algorithms to more efficiently search of agent invariants. Additionaly, future plans include running further experiments with various number of generations and population sizes to carry out a finer-grained analysis on the impact of the exploration/exploitation processes.

VII. Acknowledgments

This research is supported by EPSRC grant EP/G009600/1 (Automated Discovery of Emergent Misbehaviour).

References

[1] Deb, K. and Pratap, A. and Agarwal, S. and Meyarivan, T., *IEEE Transactions on Evolutionary Computation, A fast and elitist multiobjective genetic algorithm: NSGA-II*,2,182-197,6,2002.

[2] Chang, G. and Roth, C. B. and Reyes, C. L. and Pornillos, O. and Chen, Y.-J. and Chen, A. P.,5807,Science,1875, *Retraction of Pornillos et al., Science 310 (5756) 1950-1953. Retraction of Reyes and Chang, Science 308 (5724) 1028-1031. Retraction of Chang and Roth, Science 293 (5536) 1793-1800*,314,2006.

[3] L. Hatton, *IEEE Computational Science and Engineering*,2738, *The T experiments: errors in scientific software*, 4,1997.

[4] K. Simons, *New England Economic Review*,17-28, *Model error - evaluation of various finance models*,1997.

[5] Holland, J. H. and Miller, J. H.,The American Economic Review, 2, 365-370, *Artificial Adaptive Agents in Economic Theory*, 81, 1991.

[6] M. Kiran and S. Coakley and N. Walkinshaw and P. McMinn and M. Holcombe, *Biosystems, Validation and Discovery from Computational Biology Models*,93,141-150,1-2,2008.

[7] Ernst, M. D. and Perkins, J. H. and Guo, P. J. and McCamant, S. and Pacheco, C. and Tschantz, M. S. and Xiao, C, *Science of Computer Programming, The Daikon system for dynamic detection of likely invariants*,2007.

[8] FLAME: Flexible Large-scale Agent-based Modelling Environment, http://www.flame.ac.uk.

[9] Sun, T. and McMinn, P. and Coakley, S. and Holcombe, M. and Smallwood, R. and MacNeil, S.,17,*Journal of the Royal Society Interface*,1077-1092,*An Integrated Systems Biology Approach to Understanding the Rules of Keratinocyte Colony Formation*,4,2007.

[10] B. Chan and J. Denzinger and D. Gates and K. Loose and J. Buchanan, *Proc. of the Congress on Evolutionary Computation, CEC2004*,125–132,*Evolutionary Behavior Testing of Commercial Computer Games*,2004.

[11] J. Denzinger and J. Kidney,*Proc. of the International Conference on Intelligent Agent Technology (IAT)*,23–29, *Evaluating Different Genetic Operators in the Testing for Unwanted Emmergent Behavior using Evolutionary Learning of Behavior*,2006.

[12] J. Kidney and J. Denzinger,*Proc. of the 17th European Conference on Artificial Intelligence (ECAI)*,260–264, *Testing the Limits of Emergent Behavior in MAS using Learning of Cooperative Behavior*,2006.

[13] Cu D. Nguyen and S. Miles and A. Perini and P. Tonella and M. Harman and M. Luck,*Proc. of AAMAS 2009*,521–528,*Evolutionary Testing of Autonomous Software Agents*,2009.

[14] J. C. Bongard and H. Lipson,*IEEE Transactions on Evolutionary Computation*,4,361 - 384,*Nonlinear System Identification Using Coevolution of Models and Tests*,9,2005.

[15] F. Klügl,*Proc. of the 2008 ACM symposium on Applied computing.*,39-43,*A Validation Methodology for Agent-Based Simulations*,2008.

[16] M. Niazi and A. Hussain and M. Kolberg, *Proc. of the Third Workshop on Multi-Agent Systems and Simulation*, *Verification and Validation of Agent-Based Simulations using the VOMAS approach*,2009.

[17] Zhiyong Zhang and John Thangarajah and Lin Padgham, *Proc. of the 2nd International Working Conference on Evaluation of Novel Approaches to Software Engineering*, *AUTOMATED UNIT TESTING FOR AGENT SYSTEMS*,2007.

[18] Roberta Coelho and Uir Kulesza and Arndt von Staa and Carlos Lucena, *Proc. of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, 83-90, *Unit testing in multi-agent systems using mock agents and aspects*,2006.

[19] Salem Adra and Phil McMinn, *Proc. of 5th International Workshop on Mutation Analysis (Mutation 2010)*, *Mutation Operators for Agent-Based Models*, 2010.