Distributed, Automated Calibration of Agent-based Model Parameters and Agent Behaviors

Extended Abstract

Matteo D'Auria
Università degli Studi di Salerno
Fisciano, Italy
matdauria@unisa.it

Eric O. Scott
George Mason University
Washington, D.C.
escott8@gmu.edu

Rajdeep Singh Lather
George Mason University
Washington, D.C.
rlather@gmu.edu

Javier Hilty
George Mason University
Washington, D.C.
jhilty2@gmu.edu

Sean Luke
George Mason University
Washington, D.C.
sean@cs.gmu.edu

ABSTRACT

Agent-based models can present special challenges to model calibration due in part to their high parameter count, tunable agent behaviors, complex emergent macrophenomena, and potentially long runtimes. However, due to this difficulty, these models are most often calibrated by hand, or with hand-coded optimization tools customized per-problem if at all. As simulations increase in complexity, we will require general-purpose, distributed model calibration tools tailored for the needs of agent-based models. In this paper, we present the results of a system we have developed which combines two popular tools, the MASON agent-based modeling toolkit, and the ECJ evolutionary optimization library. This system distributes the model calibration task over many processors, provides many stochastic optimization algorithms well suited to the calibration needs of agent-based models, and offers the ability to optimize not just model parameters but agent behaviors.

KEYWORDS

Agent-based Models, Model Calibration, Evolutionary Computation

ACM Reference Format:
Matteo D'Auria, Eric O. Scott, Rajdeep Singh Lather, Javier Hilty, and Sean Luke. 2020. Distributed, Automated Calibration of Agent-based Model Parameters and Agent Behaviors. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 3 pages.

1 INTRODUCTION

Agent-based models (ABMs) are widely used in research areas such as computational biology, the social sciences, multi-agent systems, and robotics. One of the most challenging steps in the development of an ABM is the calibration of the model's parameters and agent behaviors. ABM calibration is challenging due to the number of agents, their heterogeneity, and the complex interaction among them. Such models are also often slow and consequently the number of possible calibration trials can be low. Because of these difficulties, many ABMs are not calibrated at all or ad hoc solutions are applied. For example, in Heppenstall et al. [1] half of the examined AMBs do not show any sort of calibration.

In this paper we address the need for a general tool that allows for the calibration and optimization of ABMs in an assisted, automatic, and distributed fashion. Given the importance and difficulty of calibrating agent-based models, this is an important item: yet there are surprisingly few existing facilities available at this time. We then demonstrate some basic capabilities of the system in distributed model optimization and in optimization of agent behaviors.

2 APPROACH

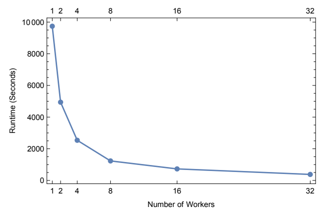
Calibration is essentially an optimization process, but as testing is done in simulation, we do not have a formal objective function nor its gradient; thus we must rely on stochastic optimization techniqies. To do ABM calibration, we have combined two tools popular in their respective fields: the MASON agent-based simulation toolkit [4] and the ECJ evolutionary optimization library [6]. The methodology is not reliant on either of them: but both make it easy to implement.

The calibration process is as follows. The modeler first builds the simulation and sets those parameter values he knows or assumes to be correct. The model is then given to the calibration system, which optimizes the parameters, or behaviors, according to an objective function specified by the modeler. The modeler then examines the results obtained, and based on these results, determines if there are errors in the model, incorrect assumptions, etc. The modeler then improves the simulation and fixed parameter values and resubmits the simulation to the calibration system in the next iteration.

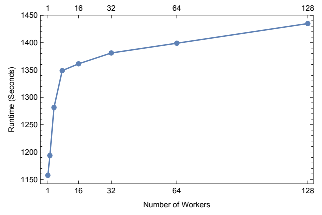
In our distributed and assisted calibration approach, the modeler must define an ECJ process called the *master*, which runs a top-level model optimization algorithm. This process maintains a *population* of candidate solutions (or *individuals*), which are tested by packaging them and sending them to remote processes called *workers*. Each candidate solution is simply a set of parameter values or behaviors which define a possible complete model definition. Each worker creates a MASON simulation using these parameters, runs it several times, and evaluates its performance. Then the worker returns the performance results to the master, which uses them in its resampling procedure on the population.

By default the modeler need only provide the ranges of parameters to be optimized. But if the modeler has special needs (a special

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.



(a) Strong Scalability



(b) Weak Scalability

Figure 1: Scalability Analysis, Refugee model

representation of agent behaviors, for example), he can completely customize the optimization procedure using ECJ, but it will require more work. ECJ has a great many stochastic optimization facilities available to use. We discuss this latter scenario in Section 3.

3 EXPERIMENTS

Our experiments were performed using a cluster of 24 machines. Each node had a Dual Intel Xeon E5-2670 @ 2.60GHz with 24 GB RAM and an Intel 82575EB Gigabit Ethernet network adapter. On each node we installed Red Hat Enterprise Linux Server 7.7 (Maipo) and OpenJDK 1.8.0. All of the experimental results were statistically significantly different from one another ($p < 0.01$) as verified by a one-way ANOVA with a Bonferroni post-hoc test. These methods use evolutionary algorithms: the details of these algorithms are described in more detail in [3].

Speedup Demonstration. For this demonstration we used the *Refugee* model, part of the collection of contributed models in the GeoMASON distribution,[1] and which explores the pattern of migration in the Syrian refugee crisis. We calibrated this model over four real-valued parameters (so-called DangerCare, FamilyAbroadCare, EconomyCare, and PopulationCare), each ranging [0...1], and compared the number of arrivals in each city in the model against real-world data. The model ran for 10,000 steps. We used a generational genetic algorithm with a tournament selection of size 2, one-point crossover, and Gaussian mutation with a 100% probability and a standard deviation of 0.01. The results are shown in Figure 1.

Strong scalability analysis: we fixed the problem to ten generations, each with 32 individuals. In our case, the strong scalability efficiency came to 71.88%, using 32 workers to solve the problem.

Weak scalability analysis: we varied the problem difficulty by adjusting the population size such that, regardless of the number of workers, each worker was responsible for four individuals per generation. For each optimization process the number of generations was fixed to 10 and the population size varied in $\{4, 8, 16, 32, 64, 128, 256, 512\}$, and thus the number of workers varied as $p \in \{1, 2, 4, 8, 16, 32, 64, 128\}$. In our case, the weak scalability efficiency was 83.18%.

Asynchonous Evolution Experiment. In this experiment we considered an *asynchronous* evolutionary algorithm (that is, the distributed version of a so-called *steady-state* algorithm) to improve efficiency when the model runtimes varied greatly. We compared the generational genetic algorithm discussed earlier against an asynchronous evolutionary algorithm using a standard steady-state genetic algorithm as its foundation. There were 128 workers and the population size was set to 128. To simulate varying runtimes in the Refugee model, we randomly varied the number of simulation steps each time: 1/4 the time we halved the steps and half the time we doubled them. The results were as follows:

Method	*Mean Runtime (Seconds)*
Asynchronous Evolution	293.77
Generational Evolution	437.77

Optimizing Agent Behaviors. Agent behaviors are essentially programs that dictate how the agents operate in the environment and interact with one another. This is more complex than simple parameter optimization, because when calibrating agent behaviors the modeler must specify both the nature of the representation of these agent behaviors, and must also write the simulation code which, when given an individual, evaluates its behaviors in the simulation proper.

An agent behavior may take different representations, such as neural networks or automata, but for demonstration we will focus here on the classic "Koza style" *genetic programming* style, where behaviors take the form of executable Lisp parse trees [2].

Our example is drawn from the *Serengeti* model [5], in which four "lion" agents are tasked to capture a "gazelle" in a real-valued toroidal environment. Each individual consists of four parse trees, one per lion. We used a genetic-programming facility closely following the approach in Luke and Spector [5]. We ran the genetic programming algorithm as described, but with a population size of 5760 spread over 276 workers: each worker was thus assigned 20 individuals per generation. Assessment of an individual's behaviors was done over 10 random trials. The results are shown in Figure 2.

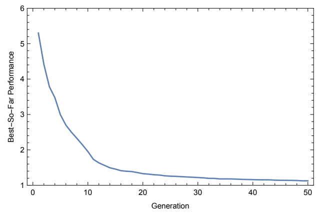


Figure 2: Mean best-so-far performance, over 30 runs, of genetic programming on the Serengeti model (lower is better).

4 CONCLUSION

Like all models, calibration of ABMs is very important, but as ABMs continue to increase in complexity and run length, model calibration becomes more and more difficult to do. ABM calibration is also unusual because agent behaviors, as well as global or agent parameters, may be optimized during the calibration process. These and other reasons motivate the use of massively distributed evolutionary optimization tools aimed at agent-based model calibration.

We have developed a tool of this kind which combines MASON and ECJ, both popular libraries in their respective fields. We have shown how their combination can produce a powerful, fully-featured model calibration facility with special capabilities of interest to the agent-based modeler. This facility will soon be available as open source.

[1] http://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/

REFERENCES

[1] Allison Heppenstall, Nick Malleson, and Andrew Crooks. 2016. "Space, the Final Frontier": How Good are Agent-Based Models at Simulating Individuals and Space in Cities? *Systems* 4, 1 (2016).

[2] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[3] Sean Luke. 2013. *Essentials of Metaheuristics* (second ed.). Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[4] Sean Luke, Robert Simon, Andrew Crooks, Haoliang Wang, Ermo Wei, David Freelan, Carmine Spagnuolo, Vittorio Scarano, Gennaro Cordasco, and Claudio Cioffi-Revilla. 2018. The MASON Simulation Toolkit: Past, Present, and Future. In *International Workshop on Multi-Agent-Based Simulation (MABS)*.

[5] Sean Luke and Lee Spector. 1996. Evolving Teamwork and Coordination with Genetic Programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*. 141–149.

[6] Eric Scott and Sean Luke. 2019. ECJ at 20: Toward a General Metaheuristics Toolkit. In *GECCO '19 Companion*.