MABE 2.0

an introduction to MABE and a road map for the future of MABE development

Clifford Bohm
Michigan State University
East Lansing, Michigan
cliff@msu.edu

Alexander Lalejini
Michigan State University
East Lansing, Michigan
lalejini@msu.edu

Jory Schossau
Michigan State University
East Lansing, Michigan
jory@msu.edu

Charles Ofria
Michigan State University
East Lansing, Michigan
ofria@msu.edu

ABSTRACT

MABE (Modular Agent-based Evolver) is an open-source evolutionary computation (EC) research platform designed to be used by biologists, engineers, computer scientists, and other researchers. MABE's primary goal is to reduce the time between thinking up a new hypothesis and generating results. The design assumes that there are common elements in many EC research projects. MABE improves efficiency by allowing for the reuse of these common elements and standardizing of interfaces for non-common elements so that they can be used interchangeably. As of the writing of this paper, the MABE framework is five years old. Here, we reflect on the current version of MABE, including its successes and shortcomings, and propose upgrades for the next release.

CCS CONCEPTS

• **Software and its engineering → Development frameworks and environments**; *Software implementation planning*; *Software post-development issues*; • **Computing methodologies** → *Artificial life*;

KEYWORDS

Modular Agent-based Evolver, MABE, Empirical Library, evolution, evolutionary computation, artificial life, open source, software development

ACM Reference Format:
Clifford Bohm, Alexander Lalejini, Jory Schossau, and Charles Ofria. 2019. MABE 2.0: an introduction to MABE and a road map for the future of MABE development. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3319619.3326825

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326825

1 INTRODUCTION

MABE (Modular Agent-based Evolver) is an open-source evolutionary computation (EC) research platform designed to be used by biologists, engineers, computer scientists, and other researchers [1]. MABE allows users to create experiments by combining five types of module components: *worlds* (problems or environments), *genomes* (sources of heritable and mutable data), *brains* (neural/cognitive digital architectures such as artificial neural networks or genetic programs), *optimizers* (parent-selection and population-management processes), and *archivists* (data tracking and recording). Modules are described in detail below. Modules of a given type are interchangeable, so switching from one brain type to another is as simple as changing a brain type parameter from "ANN" to "GP".

MABE is implemented using C++14 (depending only on standard libraries). MABE source code and documentation can be accessed at https://github.com/Hintzelab/MABE.

MABE was inspired by the observation that many EC programs share common concepts (*e.g.*, fitness functions, selection schemes, populations, *etc.*). A system that leverages these similarities would, not only allow for efficient reuse of common components, but also remove communication road blocks and simplify comparison, replication, and integration of results that would otherwise be generated by different systems. Of course, there is a reason that researchers use a diversity of software: different EC systems have unique behaviors and are required to work in different ways. **Our goal was to design a system that provides sufficient structure to support portable, reusable, and interoperable components, while providing the flexibility needed to implement different EC systems.**

MABE is a software framework that improves efficiency though component reuse in order to reduce the time between thinking up a new hypothesis and generating results. In addition, MABE promotes synergistic research by allowing users from different backgrounds to work in a single software ecosystem.

MABE is designed to be used by individuals with a range of programming skills and academic backgrounds (*e.g.*, evolutionary and ecological biology, neural biology, economics, psychology, engineering, computer science, evolutionary computation, *etc.*). MABE affords non-programmers an off-the-shelf software platform with an ever-growing collection of ready-made modules that can be combined and configured using a parameters system that requires no interaction with code. More advanced programmers can develop

their own modules to extend MABE's functionality: new modules are immediately compatible with all previously existing modules, and each new module expands the set of possible systems that can be constructed using MABE.

Modules allow users to focus their efforts on topics important to them without requiring a detailed understanding of the entirety of MABE. For example, an evolutionary computation researcher interested in determining how selection methods affect rates of adaptation could write an optimizer that allowed them to investigate different hypotheses. They would not need to worry about any other code, as they could make use of exiting modules and core functions for the rest of their research system.

MABE is a living software project. EC is a fast-paced field, so in order for MABE to keep up we are always implementing new ideas. In order to balance stability for existing users and potential instability generated from integrating new features, we rely on version control methods and versioned releases. Most new releases are designed to avoid breaking changes (that is, they either add functionality or fix bugs but do not change existing behavior). On the rare occasion that we do need to make a breaking change, we provide information about how the changes will affect current users and provide support for upgrading to the newest version. On very rare occasions, we may feel that accumulated small changes or support for new features requires us to revisit the basic structure of MABE, and this may result in significant system-wide changes.

MABE is in its fifth year, and we have decided that the first significant system-wide rewrite of the software is warranted. The various reasons for the rewrite are discussed in detail later in this paper. We are currently in the process of designing the next iteration of MABE (version 2.0). As we design and implement MABE 2.0 (and moving forward), we are broadly seeking community feedback on the current version, and plans for the new version, as well as suggestions for features beyond the scope the current redesign.

In this paper, we briefly talk about who is using MABE and for what. Then, we provide an introduction to MABE as it exists now including some more in-depth analysis of design choices than was investigated in previous publications. This is followed by a section describing proposed changes and additions for MABE 2.0. For a more basic introduction to MABE see [1].

1.1 Who is using MABE and for what?

MABE has been used to conduct a broad array of computational evolution research, including evolved cognition [25], decision making [15], internal representation [10, 17], learning [27, 28], swarm behavior [5], psychology [22], and hybrid computational architectures [11]. MABE's modular, mix-and-match design and a broad set of existing MABE modules [1] has reduced the barrier to carrying out large-scale experiments that would otherwise be unwieldy.

In one paper that exemplifies the advantage of MABE's modular design, the Buffet Method, proposed by Hintze *et al.* [11] capitalized on MABEs ability to swap worlds. This work extended Markov Brains [9] by adding new gate types (computational elements) representative of other EC systems (GP trees, NEAT [30] and, ANNs), allowing these elements to interact directly in a single network representation. Evolution was able to compose heterogeneous networks from arbitrary combinations of these elements. To explore

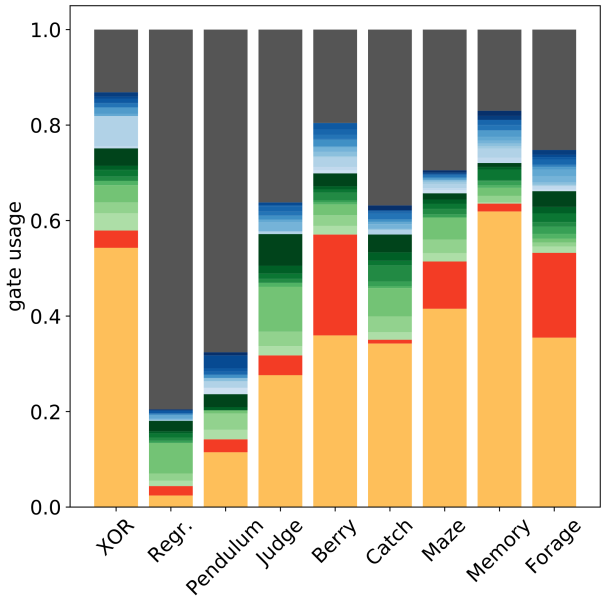

Figure 1: Data generated using MABE showing normalized gate usage for Markov Brains evolved in different worlds. Brains were initialized with multiple types of randomly configured gates, the ratios of which changed over time as a result of evolution. The x axis lists the worlds that were tested and the y axis shows gate usage by gate type recorded from brains after 5,000 generations. Colors indicate gate type: classic Markov Brain gates (in orange: deterministic logic, in red: probabilistic logic), Genetic Programming gates (in shades of green), NEAT gates (in shades of blue), and ANN gates (in dark grey). The data illustrate that Deterministic Markov Brain gates were favored in the XOR and Memory worlds, whereas ANN gates were favored in the Regression and Pendulum worlds. This figure was generated using Python and Matplotlib.

the efficacy of the Buffet Method, Hintze *et al.* [11] evaluated the 'Buffet Method' networks (*brains*) in nine different worlds that defined tasks from various EC domains. Homogeneous networks, comprised of a single element type, were well-suited for solving some, but not all, of the problems. However, by giving evolution access to all of these computational elements simultaneously, the Buffet Method discovered high performance (often hybrid) solutions across all problems. Figure 1 shows how the evolved solutions for different tasks differed in gate usage, suggesting that different types of computational elements are beneficial in different situations and that evolution can be used as a method to detect these differences.

In other work, Hintze et al. [10] used MABE to study *representation* or "R" (a measure of how much environmental information from previous observations is stored internally) and *smearedness* (how focused or distributed the environmental information is among the elements that make up a brain) by evolving three

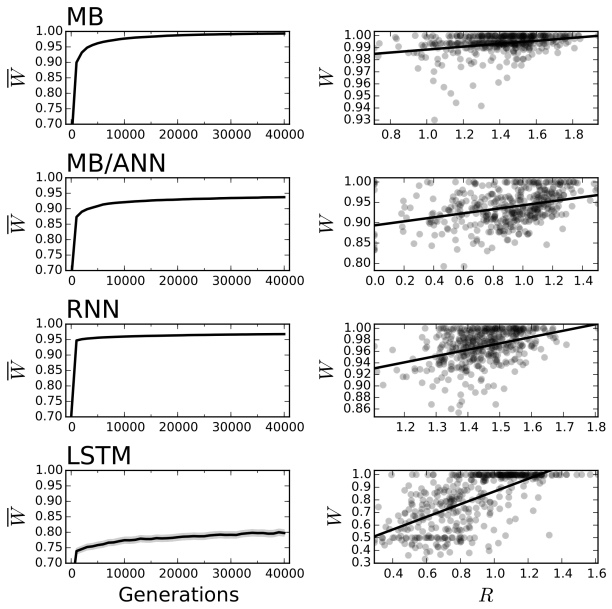

Figure 2: Data generated using MABE to evolve agents preforming a quantity comparison task. The left column shows mean fitness (W), reported as a percentage of trials that the agents answered correctly, averaged over 400 replicate experiments. Standard error is shown in grey. The right column displays fitness versus representation (R) at 40,000 generations for each replicate experiment where representation is a measure of information shared between a brain and the environment. The black line in the right column is the line of best fit. Each row represents data from a different type of brain. It can be observed, for example, that Markov Brain (MB) / ANN hybrid brains and RNN brains both achieve similar fitness but that the levels of representation are higher and have less variance in RNN brains. This figure was generated using Python and Matplotlib.

different types of brains (long short-term memory units (LSTM), recurrent neural networks (RNN), and Markov brains) in a block catching environment (for a description of "R" and the environment see [18]). In more recent work, the same team has looked at how four types of brains (LSTM, RNN, Markov Brains with ANN gates, and Markov Brains with deterministic logic gates) evolved in a world where agents were required to use memory on a value comparison task, and they showed how noise in inputs affected "R" and smearedness across different brain types[14]. Figure 2 shows scores ("W") over time and final score vs representation ("R") for each replicate for various types of brains in the absence of input noise (the control case).

Beyond published research, MABE has been used by students in both undergraduate and graduate classes at Michigan State University: "Multi-disciplinary Approaches to the Study of Evolution", "Digital Approaches to the Evolution of Nervous Systems", "Evolution of Artificial Intelligence", and "Evolutionary Computation". MABE allowed students to focus on a single aspect of their project, developing or modifying the modules of interest and reusing existing modules when possible. In the context of the classroom, MABE's capacity for module-reuse coupled with a repository of existing (and well-vetted) module implementations allowed students to tackle more challenging projects than would have otherwise been feasible.

2 MABE DESIGN

Many evolutionary computation experiments share common concepts, such as a population of individuals (*e.g.*, programs, neural networks, candidate solutions, simulated biological organisms, *etc.*) where each individual is encoded (specified) by mutable and heritable material, an environment or problem of interest which may be used directly or indirectly to evaluate agents, parent-selection and population management techniques, data tracking and recording functions, and user-specified parameters. The implementations of some of these concepts can be shared across experiments without modifications, while the need for different behaviors in other concepts may require changes in implementation from experiment to experiment. For example, different users may need customized implementations for genetic encodings, memory representations, or parent-selection techniques, but the functions allowing for parameter specification, data recording, and random number generation need not change because of the experiment.

The concepts that can be held constant across all (or most) experiments are part of *core*. Core includes functions to initiate and execute experiments and *utilities* that provide general support for parameters, data management, file I/O, and random number generation. Any concept that cannot be held constant (*i.e.*, requiring unique internal implementation) must be part of a module. MABE core defines the interfaces for each type of module (*i.e.*, how the module communicates with MABE and other modules), but in order to provide the highest degree of flexibly, the module interfaces do not define any specific details related to a module's internal operations. The implementation of modules is left entirely to users. As long as a user defined module conforms to the module interface, then there are no restrictions on how a module behaves. MABE's design rests on the foundation that we can provide standard interfaces to modules without limiting their internal algorithms, data structures, or implementation details. This design affords considerable experiment decomposability, allowing researchers to share and reuse individual modules across experiments.

We identified five concepts that needed to be modules in order to support the the variety of experiments possible in MABE: worlds, genomes, brains, optimizers, and archivists. In figure 3 we illustrate how the modules typically interact.

- **Worlds** are problems or environments. Generally, worlds provide inputs (*e.g.*, the current world state) to agents and respond to outputs generated by agents. A simple world may deliver two numbers to an agent and evaluate the agent's ability to add those numbers. Some worlds provide no inputs; for instance, a multi-objective optimization problem or a fitness function like an NK fitness landscape [12] may not have world state, and only require agents to deliver a list of output values. Finally, a complex world could be designed where agents must navigate a virtual space, foraging for food

while avoiding predators and seeking mates. The input might communicate aspects of world state, such as the relative positions of food and danger, and the agents outputs could be interpreted by the world as actions such as turn left or right, move forward or back, eat, *et cetera*.

- **Genomes** are sources of heritable and mutable data. Genomes are usually used to provide data used to construct other elements: a genome may be used by a brain to determine that brain's structure or by a world to determine the properties of an agent's body. The genome interface requires access functions to write values into the genome (including randomizing the genome) and read values from the genome. Each type of genome defines its own internal data structures, access function behaviors, and mutation operators.
- **Brains** are data processors that receive input and deliver output. Brains are the most common method by which agents communicate with worlds (although worlds interfacing directly with genomes is an option). Under this abstraction, genetic programs, artificial neural networks, Markov Brains, *et cetera* are classified as brains. Depending on the user's context, they may find it easier think of brains as controllers, solvers, solutions, I/O machines, or even chemical processes. The brain interface specifies how brains can receive input and deliver output. Each type of brain must define its internal workings, including update function, internal states, and internal data structures.
- **Optimizers** manage populations, select parents, oversee reproduction, and terminate agents who get to be too old or fail to meet certain criteria. Roulette selection, tournament selection, lambda+n, lexicase selection [8], and MAP-Elites [19] are all examples of the types of algorithms optimizers typically implement.
- **Archivists** allow users to determine what data will be saved and at what resolution. Archivists can save individual or population level data and can track lines of decent and other temporal effects.

The typical experiment uses one module of each type, but this is not a requirement. Consider a foraging world where organisms divide whenever they collect enough resources. This world might manage reproduction locally and not rely on an optimizer. Or, consider a world where agents have a brain and world-defined sensors that are are placed using a genome. A user could configure MABE so that the agents had a single genome that was used to generate both the brain and sensor placement (allowing for genetic interactions), or the user could configure MABE so that the agent had two genomes (in which case the brain and sensor placement would be genetically independent).

3 DESIGN PHILOSOPHY: WHAT BELONGS IN CORE? WHAT BELONGS IN MODULES? IS THERE A PLACE FOR LIBRARIES?

We have been deliberate to only include concepts in core that have broad utility and can be generalized across experiments. Attempting to create standards for elements that cannot be generalized (such as how vision functions in a world - see below) may be useful to some users but would create road blocks for users who require

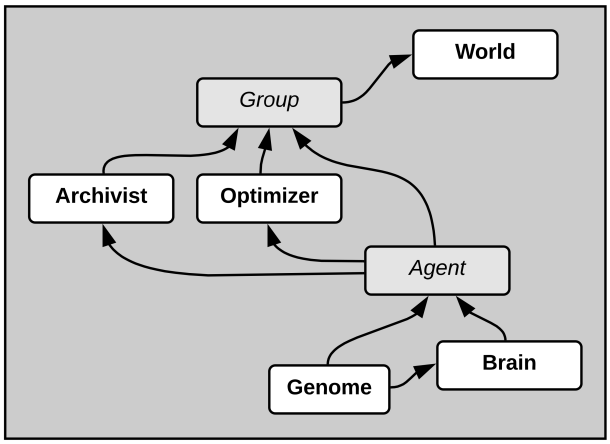

Figure 3: A simple diagram of the organization of MABE's modules in a typical configuration. World, Genome, Brain, Optimizer and Archivist are Modules. *Agent* and *Group* are internal container classes.

behavior not supported by the standard. In some cases, we have been able to identify core features that will be of use to some users while not hindering the remainder of users; in these cases, we may have decided to include the feature if it is either trivial (*i.e.*, will not degrade efficiency or contribute significantly to code bloat) or if the feature is useful to a significant proportion of the user base.

Much of the difficulty in designing MABE has been determining when to standardize (*i.e.*, include in core) and when to insist that some element must be user defined (*i.e.*, be specified in a module). While some choices are obvious, other choices are complex, and even after significant deliberation it can be non-trivial to decide whether or not a feature should be part of core. The following examples are case studies related to determining when a particular functionality should be core and when it should not. The first example is a case addressing brain-world communication where functionality was included in core. This is followed by two cases, relating to brains' internal data and implementing worlds, where functionality was not added to core.

Example 1. The brain module interface definition lists the functions that must be part of all brains. Three such required functions define behavior to a) set brain inputs (a fixed number of real number values), b) update the brain (*i.e.*, trigger some process defined by the brain to convert inputs into outputs), and c) read the resulting outputs (also a fixed number of real number values). But, not all brains need all of the functionality specified in the brain interface definition. One brain type, 'ConstantValueBrain', converts a genome to a fixed list of output values. This brain requires neither input nor update because the brain determines its outputs on construction. This brain is perfectly capable in some worlds (max one, NK fitness landscapes), so it is reasonable that a user would want to use it. We could have decided that the brain interface should only be required to provide output values, but most brains cannot be represented by a static list; they accept inputs and respond differently to different inputs. We therefore decided to include the "set input" and "update"

functionality in the brain interface definition. As a result, when writing "ConstantValueBrain", these functions need to be present (but can be empty). We see this as a small and rare overhead cost important to achieving modularity.

Example 2. On the other hand, many, but not all brains, have internal (or hidden) information. This hidden information is required for stateful brains and memory. Unlike the input and update functions discussed above, we cannot include functionality related to a brain's hidden information in the brain interface definition because, we cannot know the data type or structure of hidden information that will be needed within a specific user-defined brain type. As a result, we cannot define functions that access hidden information in the brain interface definition without the risk of limiting the types of brains that can be built in MABE.

Example 3. Finally, let us consider the comparison of brain-world communication (functionality defined in core) versus vision in a world (functionality a user must provide). MABE standardizes brain-world communication via two lists of numbers (the brain input and output vectors described above). This I/O vector design supports many architectures (*e.g.*, Markov Brains, ANN, GP, CGP) and allows users to create many worlds and brains and to connect them arbitrarily. On the other hand, vision is intertwined with the definition of the world (is the world two-dimensional or three-dimensional? What is the depth and width of the view arc? What is the resolution of the view sensor? Are there colors? Are the vision sensors perfect or imperfect? If imperfect, what sort of error?). Thus, vision must be left to the world builder to define, or we risk creating a scenario where a user is not able to conduct a particular experiment without resorting to workarounds.

MABE does not preclude the use of third-party libraries in user-generated modules; some experiments may require specialized (non-generalizable) functionality, requiring users to provide their own solutions. In fact, using third party libraries is advisable when possible and particularly when it allows for code reuse. Libraries have been created to support various areas that may be of use to module development such as physics simulations (Bullet [2], Chrono [33]), robot control [23], *et cetera*. One of MABE's strengths is that it places no restrictions on module-level code, so external systems and libraries can be incorporated into user-developed modules. In MABE 2.0 documentation, we will include suggestions for third-party libraries that can be used to support common tasks in module building and, when appropriate, we will even warehouse libraries if they prove to be particularly useful.

4 MABE 2.0: THE FUTURE OF MABE

MABE originally started as a proof of concept and has developed into a capable research tool. Over MABE's four years of development, we have been able to investigate how early choices in design affected the resulting software. We have also experienced using MABE first-hand and have collected feedback from other users. We have identified a number of areas for improvement and are in the process of preparing a new major release. Over the second half of 2018, we held a series of meetings, including individuals outside the project, to investigate potential design upgrades for MABE 2.0. The focus of these meetings was two-fold: (1) how best to improve on the current features of MABE, and (2) how to expand the audience while avoiding generalization traps (*i.e.*, how can we add features that do not create restrictions?). We have also joined with the developers of Empirical [20], a software library that provides tools for developing high-performance scientific software in C++ that is also web-capable. In MABE 2.0, Empirical library components will be used in core development and will be available to module builders. The remainder of this paper is dedicated to discussing the proposed improvements and upgrades that we plan to implement for MABE 2.0.

4.1 The Empirical library and web integration

The Empirical library [20] provides a set of well-tested software tools designed to streamline development of *efficient* scientific software. These tools include configuration management, signalling techniques, data handling, and numerous customized data structures and utilities (*e.g.*, fast functions for pulling random numbers from a binomial distribution).

Empirical is built to facilitate use of the Emscripten compiler [37], which compiles C++ to Web Assembly and allows for web applications the run at near-native speeds. Empirical includes C++ functions powered by Emscripten that provide building blocks supporting the development of interactive web based visualizations that will run on any web browser.

MABE 2.0 will be built using the Empirical library, and Empirical will ship with MABE allowing module developers to easily make use of tools that it provides. Furthermore, putting MABE 2.0 onto the web will allow end users to trivially run the software without needing to manually download or compile on their own computer. The web support that Empirical and Emscripten provide will allow MABE 2.0 to automatically generate intuitive graphical interfaces augmenting the current text only configuration files. We will adhere to the HTML5 standard to ensure that no additional plugins are required to use the software. Web-enabled software has the additional benefits of allowing for easy integration of applications into blog posts or JavaScript enabled presentations, and these applications can be combined with existing JavaScript packages to provide high-end user experiences.

4.2 Non-linear code flow

The current version of MABE combines object-based design (particularly in module structure) and procedural design (to control high level program flow) [6]. As a result, module definitions are quite flexible, but high-level execution (the order of operations such as agent evaluation, reproduction and death, and when data is archived) is restrictive. This was an intentional choice to accept less flexibility for more modularity. In order to allow for more complex flow control, while still maintaining modularity, we plan to implement a signal system that can trigger the execution of code based on certain defined events. A signaling system would provide a structured set of rules that would allow module developers to break out of the prescribed procedural flow by introducing user-defined events and the code that will be executed when those events occur. In addition, core could define signals, such as "onBirth", "onDeath", "newGeneration", *et cetera*, allowing users the option to insert their own response when the associated event occurs.

4.3 World-brain Interaction

In the current implementation, brains and worlds interface via two access functions that handle brain I/O and an update function that executes the brain's internal code (effectively converting input values to output values). This design is simple and supports any neural architecture that a) has inputs and outputs that can be represented by lists of numbers and b) an update procedure that is always run to completion. This allows for many types of architectures (*e.g.*, Markov Brains, ANN, RNN, LSTM, Tangled Program Graph [13], GP, and CGP), but it fails to effectively cover architectures that have partial updates (*e.g.*, Avida [21] and Signal GP [16]) or that allow either partial input or require input that cannot be represented as a list of numbers (PushGP [29], convolutional ANNs). Both shortcomings could be overcome in the current design, but it would be programmatically challenging and unaesthetic.

It is noteworthy that the current model of world-brain interaction **does not** support sending data that is not formatted as a list of numbers between worlds and brains. For example, we cannot communicate 2D image data directly. We can discretize an image; however, this will discard data formatting and require the brain to evolve the ability to reconstitute the image or require the brain to be written with knowledge of the input format. Any hard-coded input-to-image conversion would make that brain incompatible with worlds that do not provide their inputs in the exact same order, breaking modularity.

In MABE 2.0 we will exchange the lists of numbers currently used for input and output for labeled signals where each signal is associated with arbitrary typed data. Worlds will be able to generate signals for brains (input) and brains will be able to generate signals for worlds (outputs). Individuals developing brain modules will need to decide on the data types that their brain will accept and then will need to implement their brain so that it can create generalized responses for the data types that they accept. Not all brains will need to be compatible with all data types. While a brain that does not have functions for working with images will not be directly compatible with a world that provides image inputs, users will be able to create meta-brains where the output of one brain can be used as the input to a second brain. For example, imagine a world in which two images containing numbers are presented to agents and the agents are expected to return the sum of the numbers displayed. A brain that can take images as input and generate a list of numbers (a "feature detection brain") could receive the inputs and then have its output directed into a Markov brain (or any of the currently supported brain types in MABE). The combination of brain composability and arbitrary data types will not only allow MABE 2.0 to implement currently unavailable brain types but it will also allow for new hybrid architectures (an area of study that shows promise [11]).

4.4 Parameters utility and data archiving

The ability for developers to easily import user-defined information is critical to the success of any research software. The current parameters system supports user-defined parameters. Each parameter includes that parameter's data type (double, int, string, bool), default value, name, category, and a usage message that appears in automatically generated configuration files. The current process for adding parameters is relatively simple: for each parameter, a developer only needs to add a variable declaration and a separate assignment that includes a function call, but this still requires changes to two files for each parameter and could be simplified. MABE automatically generates configuration files with usage information based on the modules in use and with parameters organized by category. MABE 2.0 will streamline the parameters system, while adding (1) arbitrary data types, (2) optional value ranges, (3) improved control over configuration file organization, and (4) more options for adding and formatting documentation in configuration files. These new features will also be leveraged by graphical tools that will automatically generate interactive configuration interfaces.

In current MABE, DataMaps are used to store data and allow for data communication between modules. Data export is handled mostly by a combination of DataMaps and a File Manger utility. The DataMap code will be rewritten both for improved efficiency and for expanding the number of types allowed (currently only double, int, string, and bool). Additional output file formats will also be supported (at least JSON and binary).

4.5 Machine learning support

The domain of machine learning is not well-supported by the current version of MABE. For instance, to use backpropagation of a neural network in the current implementation, a user must create a brain module that correctly backpropagates a training delta. Additionally, the user must write a world module that supplies specific input sequences that also contain training information. Unfortunately, this breaks modularity of both the brain and world. In the redesign, we plan to add functionality (most likely in the form of signal events) to brains to make them more accessible allowing machine learning optimization methods to be implemented while adhering to MABE modular design. With signals and data passing for input, output, and feedback events, brain modules will be free to process or ignore such data. This separation by signal also allows feedback phases to occur at irregular intervals should the world be designed for this. In this way, MABE 2.0 will allow seamless interaction between evolutionary methods and machine learning methods, simplifying the creation of hybrid evo-ML brains, which is of interest to the field of Auto Machine Learning (a.k.a AutoML or meta optimization algorithms [24, 31, 34]). In addition, we will include several initial World modules for standard machine learning problems: Fashion-MNIST [36], pole balancing [4, 32], half-cheetah [7, 35]. This prepackaging and modularity of problems may eventually allow MABE to be a useful benchmarking tool, especially given the ability to compare wildly different problems and computational substrates.

4.6 Improved user experience

MABE has three (mutually non exclusive) types of users: *end users*, *modules builders*, and *core developers*. For end users, MABE is a finished product that they interface with via configuration files. Beyond understanding what they are using MABE to study, these users only need to be able to set parameters and software options. Improvements to the end user experience will mostly involve better user interfaces, such as graphical systems for configuration (for example, slider controls with meaningful bounds), better organized

configuration options, parameter namespace management, documentation that is accessible from within configuration options, and interactive world and agent interfaces that display real time status. These improvements will be particularly useful in educational settings.

Module builders are users who build or modify modules (genomes, brains, worlds, optimizers, or archivists). These users must be familiar with basic C++ (variables, loops, functions, *etc.*) as well as the interface definition for the modules they are developing. Module builders will see the most improved user experience. First, the signal manager will allow for greater control over execution flow which will allow for more complex algorithms. The brain world communication upgrades will allow for more complex interfacing between brains and worlds, which will allow for new types of brains. The parameters utility improvements will simplify the process of adding parameters and managing parameters file and interface organization.

Finally, core developers decide what features are part of core and are responsible for writing core. Core developers must be fluent in C++. Since we have a much better idea of the requirements for MABE 2.0, the overall design will be more streamlined and logical. Documentation, testing, and the build system will be developed alongside the MABE 2.0 code and so will receive more attention and more integrated support then the current version.

5 CONCLUSIONS

When we began MABE development five years ago, we had an idea to create a research tool with broad applicability for a wide range of users across many domains. The code structure and feature set required to achieve this goal was not clear, and we were not confident that our goal was even possible. We now feel confident that it is! Along the way, we realized that MABE would be more than just a time-saving tool, it changed the way we thought about our research. Because MABE enabled swapping modules and encouraged investigating questions in different ways, we began to look at problems from different perspectives and to become more aware of similarities among fields that we previously viewed as isolated. Working on MABE has lead us to work on new projects and to seek out interactions with people working in fields that we once thought were quite different than our own only to find out they are asking many of the same questions. Our goal now is for MABE to become a free and open source project managed by a self-governed community of developers. This is a model that has worked for other large software projects such as the C++ and Python programming languages, Cytoscape [26] (a network visualization and analysis tool), and khmer [3] (a tool for prepossessing large DNA sequencing data sets).

We are excited to see where MABE development will lead. We are hopeful that MABE will not only accelerate research, but bridge disciplines allowing for the synthesis of new ideas that advance Evolutionary Computation and its applications.

ACKNOWLEDGMENTS

We extend our thanks to Dr. Arend Hintze and the Integrative Biology Department at Michigan State University (MSU) for providing the funding for MABE development. We would like to thank Dr. Arend Hinzte and Douglas Kirkpatrick for permission to use their data and Figures 1 and 2. This work was supported by the National Science Foundation (NSF) through the BEACON Center (Cooperative Agreement DBI-0939454) and a Graduate Research Fellowship to AL (Grant No. DGE-1424871). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF or MSU.

REFERENCES

[1] Clifford Bohm, Nitash C G, and Arend Hintze. 2017. MABE (modular agent based evolver): A framework for digital evolution research. In *Artificial Life Conference Proceedings 14*. MIT Press, 76–83.

[2] Erwin Coumans et al. 2013. Bullet physics library. *Open source: bulletphysics. org* 15, 49 (2013), 5.

[3] Michael R Crusoe, Hussien F Alameldin, Sherine Awad, Elmar Boucher, Adam Caldwell, Reed Cartwright, Amanda Charbonneau, Bede Constantinides, Greg Edvenson, Scott Fay, et al. 2015. The khmer software package: enabling efficient nucleotide sequence analysis. *F1000Research* 4 (2015).

[4] PEK Donaldson. 1960. Error decorrelation: a technique for matching a class of functions. In *Proceedings of the Third International Conference on Medical Electronics*. 173–178.

[5] Dominik Fischer, Sanaz Mostaghim, and Larissa Albantakis. 2018. How swarm size during evolution impacts the behavior, generalizability, and brain complexity of animats performing a spatial navigation task. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 77–84.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (1 ed.). Addison-Wesley Professional.

[7] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. 2015. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*. 2944–2952.

[8] Thomas Helmuth, Lee Spector, and James Matheson. 2015. Solving Uncompromising Problems With Lexicase Selection. *IEEE Transactions on Evolutionary Computation* 19, 5 (oct 2015), 630–643. https://doi.org/10.1109/TEVC.2014.2362729

[9] Arend Hintze, Jeffrey A. Edlund, Randal S. Olson, David B. Knoester, Jory Schossau, Larissa Albantakis, Ali Tehrani-Saleh, Peter Kvam, Leigh Sheneman, Heather Goldsby, Clifford Bohm, and Christoph Adami. 2017. Markov Brains: A Technical Introduction. (sep 2017). arXiv:1709.05601 http://arxiv.org/abs/1709.05601

[10] Arend Hintze, Douglas Kirkpatrick, and Christoph Adami. 2018. The structure of evolved representations across different substrates for artificial intelligence. In *Artificial Life Conference Proceedings*. MIT Press, 388–395.

[11] Arend Hintze, Jory Schossau, and Clifford Bohm. 2019. The evolutionary Buffet method. In *Genetic Programming Theory and Practice XVI*. Springer, 17–36.

[12] Stuart A. Kauffman and Edward D. Weinberger. 1989. The NK model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology* 141, 2 (nov 1989), 211–245. https://doi.org/10.1016/S0022-5193(89)80019-0

[13] Stephen Kelly and Malcolm Heywood. 2018. Emergent Tangled Program Graphs in Multi-Task Learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 5294–5298. https://doi.org/10.24963/ijcai.2018/740

[14] Douglas Kirkpatrick and Arend Hintze. 2019. The role of ambient noise in the evolution of robust mental representations in cognitive systems. In *Submitted to Artificial Life Conference Proceedings*. MIT Press.

[15] Peter D Kvam and Arend Hintze. 2018. Rewards, risks, and reaching the right strategy: Evolutionary paths from heuristics to optimal decisions. *Evolutionary Behavioral Sciences* 12, 3 (2018), 177.

[16] Alexander Lalejini and Charles Ofria. 2018. Evolving event-driven programs with SignalGP. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18*. ACM Press, New York, New York, USA, 1135–1142. https://doi.org/10.1145/3205455.3205523 arXiv:1804.05445

[17] Barbara Lundrigan, Laura Smale, and Arend Hintze. 2018. The effect of periodic changes in the fitness landscape on brain structure and function. In *Artificial Life Conference Proceedings*. MIT Press, 469–476.

[18] Lars Marstaller, Arend Hintze, and Christoph Adami. 2013. The evolution of representation in simple cognitive networks. *Neural computation* 25, 8 (2013), 2079–2107.

[19] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. (apr 2015), 1–15. arXiv:1504.04909 http://arxiv.org/abs/1504.04909

[20] Charles Ofria, Emily Dolson, Alexander Lalejini, Jake Fenton, Matthew Andres Moreno, Steven Jorgensen, Robin Miller, Jason Stredwick, Luis Zaman, Jory Schossau, Lauren Gillespie, Nitash C G, and Anya Vostinar. 2019. Empirical. (Feb.

2019). https://doi.org/10.5281/zenodo.2575607
[21] Charles Ofria and Claus O Wilke. 2004. Avida: A software platform for research in computational evolutionary biology. *Artificial life* 10, 2 (2004), 191–229.
[22] Abigail Ortiz, Kamil Bradler, and Arend Hintze. 2018. Episode forecasting in bipolar disorder: Is energy better than mood? *Bipolar disorders* 20, 5 (2018), 470–476.
[23] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.
[24] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. 2017. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041* (2017).
[25] Jory Schossau and Arend Hintze. 2018. Neuronal Variation as a Cognitive Evolutionary Adaptation. In *Artificial Life Conference Proceedings*. MIT Press, 57–58.
[26] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research* 13, 11 (2003), 2498–2504.
[27] Leigh Sheneman and Arend Hintze. 2017. Evolving autonomous learning in cognitive networks. *Scientific reports* 7, 1 (2017), 16712.
[28] Leigh Sheneman and Arend Hintze. 2017. Machine Learned Learning Machines. *arXiv preprint arXiv:1705.10201* (2017).
[29] Lee Spector and Alan Robinson. 2002. Genetic Programming and Autoconstructive Evolution with the Push Programming Language. *Genetic Programming and Evolvable Machines* 3, 1 (March 2002), 7–40. https://doi.org/10.1023/A:1014538503543
[30] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (jun 2002), 99–127. https://doi.org/10.1162/106365602320169811
[31] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.
[32] Andrew Stephenson. 1908. XX. On induced stability. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 15, 86 (1908), 233–236.
[33] Alessandro Tasora, Radu Serban, Hammad Mazhar, Arman Pazouki, Daniel Melanz, Jonathan Fleischmann, Michael Taylor, Hiroyuki Sugiyama, and Dan Negrut. 2015. Chrono: An open source multi-physics dynamics engine. In *International Conference on High Performance Computing in Science and Engineering*. Springer, 19–49.
[34] Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. ACM, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629
[35] Paweł Wawrzyński. 2009. A cat-like robot real-time learning to run. In *International Conference on Adaptive and Natural Computing Algorithms*. Springer, 380–390.
[36] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017). arXiv:cs.LG/1708.07747
[37] Alon Zakai. 2011. Emscripten: An LLVM-to-JavaScript Compiler. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA '11)*. ACM, New York, NY, USA, 301–312. https://doi.org/10.1145/2048147.2048224