Harnessing Digital Evolution

Philip McKinley, Betty H.C. Cheng, Charles Ofria, David Knoester, Benjamin Beckmann, and Heather Goldsby
Michigan State University

In digital evolution, self-replicating computer programs—digital organisms—experience mutations and selective pressures, potentially producing computational systems that, like natural organisms, adapt to their environment and protect themselves from threats. Such organisms can help guide the design of computer software.

Nearly 150 years ago, Charles Darwin explained how evolution and natural selection transformed the earliest life forms into the rich panoply of life seen today. Scientists estimate this process has been at work on Earth for at least 3.5 billion years.

But we remain at the dawn of evolution in another world: the world of computing. There, evolution helps humans solve complex problems in engineering and provides insight into the evolutionary process in nature. As computing power continues to increase, researchers and developers apply evolutionary algorithms to an ever-widening variety of problems. As the "Evolution in a Computer" sidebar shows, evolutionary computation methods such as genetic algorithms have already achieved considerable success, rivaling and surpassing human designers in problem domains as wide-ranging as flash memory sticks and aircraft wings.

We are investigating how to harness the power of evolution to help construct better computer software. The increasing interaction between computing technology and the physical world motivates this work. Systems must adapt to their environment, compensate for failures, optimize performance, and protect themselves from attacks—all with minimal human intervention.[1,2]

To design robust and resilient computational systems, we can take inspiration from nature. Living organisms have an amazing ability to adapt to changing environments, both in the short term through phenotypic plasticity and in the longer term through Darwinian evolution. Indeed, no existing cybersystem rivals the complexity of Earth's biosphere, yet life on Earth has evolved to not only deal with this complexity but to thrive on it.

Many researchers have studied how to use the characteristics of natural systems to design better computing systems. One approach mimics the behaviors of social insects and other species. However, while such biomimetic methods have shown promise in controlling fleets of unmanned robotic systems and in other applications, they can only codify behaviors observed in nature *today*. Purely biomimetic approaches seek to imitate the results of evolution, but they do not account for the process of natural selection that produced those behaviors.

For example, we can design the control software on a microrobot so that it mimics certain behaviors found in ants. However, while the robot might possess some physical characteristics reminiscent of an ant, the differences vastly outnumber the similarities. On the other hand, if we had the ability to *evolve* the control software, taking into account the capabilities of the robot and the characteristics of its environment, new behaviors might emerge that more effectively control the robot.[3]

DIGITAL PETRI DISH

Digital evolution gives us this power, and we are investigating how it can aid us in designing robust computational systems. Digital evolution is a form of evolutionary computation in which self-replicating computer

Published by the IEEE Computer Society
0018-9162/08/$25.00 © 2008 IEEE

Evolution in a Computer

Evolutionary computation,[1] a subfield of computer science, applies the basic principles of genetic evolution to problem solving. EC is based on evolutionary biology and extends into many other fields, including artificial life. In general, an EC system contains one or more populations of individuals that compete for resources in a computational environment. These individuals produce offspring according to their fitness, which often depends on the problem domain. The most well-known evolutionary computation method is the genetic algorithm.[2] In this iterative search technique the individuals in the population are encodings of candidate solutions to an optimization problem. In each generation, the fitness of every individual is calculated, and a subset of individuals is selected, recombined, or mutated, and moved to the next generation.

Genetic programming[3] provides a related method in which the individuals are actual computer programs. These approaches and other EC methods have been used to solve complex problems, in some cases producing patentable designs.[3] The annual Genetic and Evolutionary Computation Conference gives awards for human-competitive results produced by genetic and evolutionary computation (www.geneticprogramming.org/hc2007/cfe2007.html).

While the broad field of evolutionary computation has been studied extensively since the 1960s, the subfield of digital evolution is much younger. Self-replicating digital organisms can be traced to the game *Core War*, which Steen Rasmussen extended in 1990 into a system he called *Core World*. Soon after, Thomas Ray designed *Tierra*, which used a streamlined and fault-tolerant genetic language.

In 1993, Charles Ofria, Chris Adami, and C. Titus Brown began developing the Avida digital-evolution platform[4] at the California Institute of Technology. In Avida, each program lives in its own address space, unlike *Tierra*'s shared address space. This enhancement increased the power of digital evolution as an experimental tool. Avida has since been used to conduct pioneering research in the evolution of biocomplexity, with an emphasis on understanding the evolutionary design process in nature. In addition to providing a tool for biologists, digital evolution provides an open-ended search technique for problems in science and engineering.

References

1. K.A. De Jong, *Evolutionary Computation: A Unified Approach*, MIT Press, 2002.
2. J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence,* Univ. Michigan Press, 1975.
3. J.R. Koza et al., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence, Genetic Programming*, Springer, 2005.
4. C. Ofria and C.O. Wilke, "Avida: A Software Platform for Research in Computational Evolutionary Biology," *J. Artificial Life*, vol. 10, 2004, pp. 191-229.

programs evolve within a user-defined computational environment.[4] These *digital organisms* receive limited resources whose use must be carefully balanced if they are to survive. As organisms replicate, instruction-level mutations produce variation within the population. Over generations, natural selection can produce instruction sequences that can realize complex behaviors, sometimes revealing unexpected and strikingly clever strategies for solving problems.

Our work uses and extends the Avida digital evolution platform. Figure 1 depicts an Avida population and the structure of an individual organism. Each digital organism consists of a circular list of instructions—its genome—and a virtual CPU, which executes the instructions. An Avida environment comprises several cells, each of which can contain at most one organism, or *Avidian*. When an Avidian replicates, the system places the offspring in a randomly selected cell, terminating any previous inhabitant. Organisms can send messages to each other, produce and consume resources, and sense and change their environment's properties. Through these interactions, an organism can gain or lose virtual CPU cycles, affecting how fast it executes instructions.

The virtual CPU architecture used in most of our studies is simple, containing three general-purpose registers {AX, BX, CX}, two general-purpose stacks {GS, LS}, and four special-purpose heads. These heads serve as pointers into the organism's genome and resemble a traditional program counter or stack pointer. The instruction set for this virtual CPU is Turing-complete, and therefore, theoretically, it can realize any computable function. Available instructions perform basic computational tasks (addition, multiplication, and bit-shifts), control execution flow, enable communication, and allow for replication. Although the instruction set resembles a traditional assembly language, it is designed so that random mutations (inserting, deleting, or changing instructions) will always yield a syntactically correct program.

Avidians receive virtual CPU-cycle rewards for performing user-defined tasks, generally defined in terms of the organisms' externally visible behaviors—their phenotype. For example, a task might require the organism to

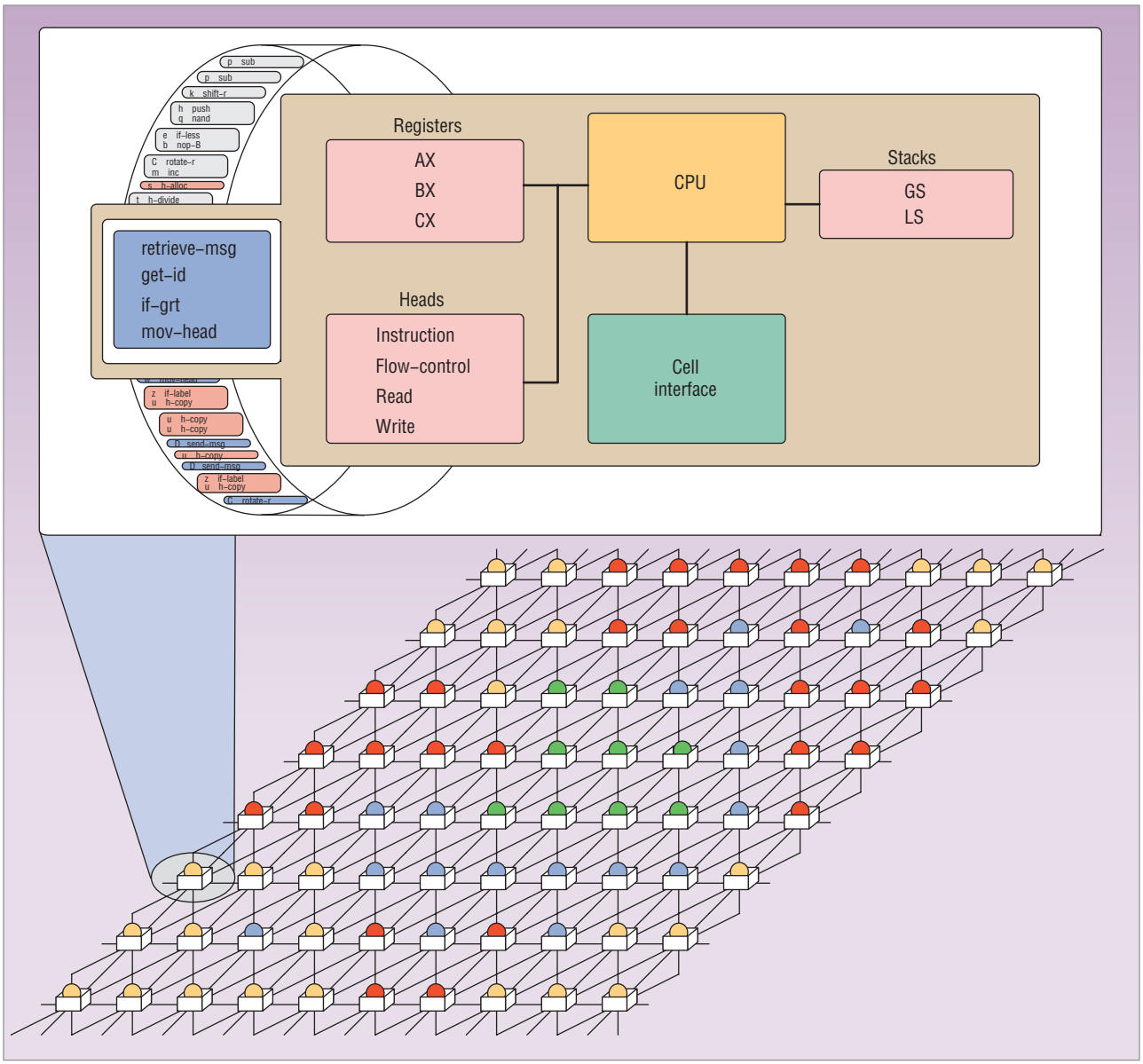

Figure 1. Elements of the Avida digital evolution platform: (top) structure of an individual organism and (bottom) a population of digital organisms in an 8 × 10 grid of cells. Different colors represent organisms with different genomes.

perform a particular mathematical or logical operation and output the result, or communicate with neighboring organisms in a manner helpful to distributed problem solving. Tasks create selective pressures in the population, favoring genomes where mutations have produced sequences of instructions that complete tasks.

In our studies, evolved code segments for completing tasks vary in length from a few instructions to several tens of instructions. An evolved solution might not be optimal when considering the task in isolation, but will likely have other properties that make it well-suited to its environment—robustness to mutation, for example. Moreover, code for completing tasks cannot mutate into the genome at the expense of replication, which is the only way organisms can pass their genetic material to future generations. Avidians that are most successful—those that replicate faster, or perform user-defined tasks—are more likely to spread throughout and eventually dominate the population. Indeed, Avida satisfies the three conditions necessary for evolution to occur[5]: replication, variation (mutation), and differential fitness (competition). Avida does not simulate evolution—it is an *instance* of evolution.

Researchers created the Avida platform primarily for studying evolution in nature. Observing evolution in digital organisms lets users address questions difficult or impossible to study with organic life forms, such as explicitly disabling selected mutations and observing the effects, or analyzing the genomes of all organisms along a particular evolutionary path. For example, Richard

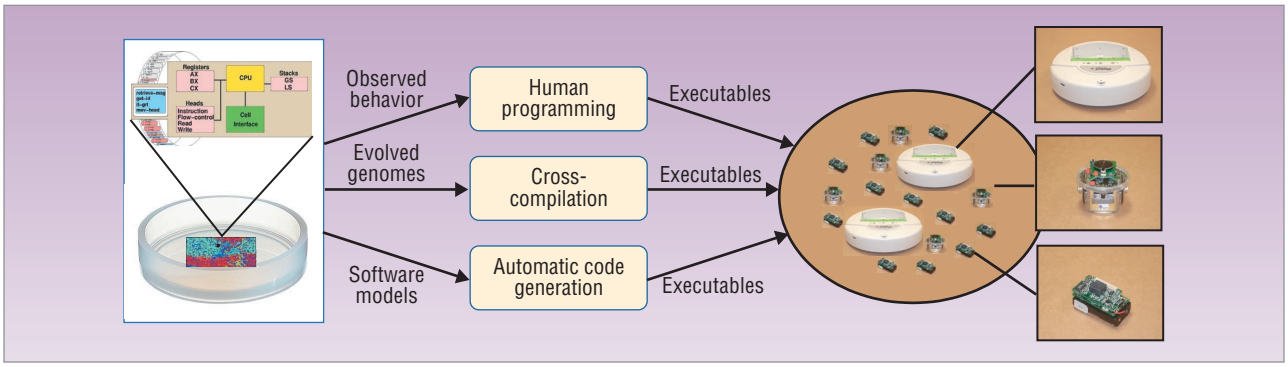

Figure 2. Different ways the authors use Avida to help develop computer software for robots and sensors. Three target platforms appear at the right of the figure and are, from top to bottom, the iRobot Create robot, e-puck educational robot, and MICA mote and sensor board.

Lenski and colleagues[6] used Avida to demonstrate that evolution can produce complex features by combining previously evolved "building blocks," helping to answer a long-standing scientific question posed by Darwin. However, Avida is an extensible platform that lets researchers and developers apply digital evolution to many different problem domains, including nonbiological ones. The user can completely customize many Avida features—including the virtual CPU architecture, instruction sets, and tasks. Today, a user with a modest compute cluster can explore hundreds of populations, totaling millions of generations, in a single day. Effectively, Avida provides the user with a digital "petri dish" for creating and analyzing new computational behaviors.

APPLICATION TO SOFTWARE DESIGN

Among other applications in science and engineering, digital evolution enables a fundamentally new approach to software design, whereby developers can actively explore new program behaviors and prospective pathways for complex software systems, all during the initial design. As depicted in Figure 2, we can apply Avida in at least three different ways to create software.

First, similar to biomimetics, behaviors that evolve in silico can provide insight into the design of new algorithms and protocols. Moreover, since digital organisms live in an environment that can be user-configured, exploration of behaviors is not limited to those found in the natural world. Novel strategies revealed through digital evolution can be codified in a traditional programming language and deployed in hardware. For example, digital evolution might yield energy-conserving behaviors that can be programmed and deployed in sensor networks.

Second, since the genomes of digital organisms are programs, they can be cross-compiled and executed directly atop hardware. For example, our group has recently developed a tool that converts Avida genomes to C code, which can be compiled and executed on a variety of devices, including sensor nodes and mobile robots. Such technologies let us test Avida-generated behaviors—such as cooperative communication operations and group-oriented mobility control—in the real world.

In the third approach, instead of evolving the software itself, we can evolve organisms that generate software artifacts. For example, we have applied digital evolution to the problem of generating and extending software design models to satisfy requirements. Specifically, Avidians act as generators and evolve to construct in-memory representations of state diagrams describing the system's behavior. Organisms can gain virtual CPU cycles by constructing state diagrams that meet requirements specified by the user, including scenarios that should be supported and properties that should be satisfied. If successful, natural selection produces a population of organisms that generate increasingly better solutions. Existing software engineering tools can be used to translate the resulting models into code.

ONGOING STUDIES

Our initial investigations focus on evolving behaviors needed in computing systems that interact with the physical world: cooperative communication, energy conservation, and adding new functionality to an existing system. When exploring a particular problem, we typically execute several batches of Avida runs, each with a different mix of tasks, then analyze the evolutionary process and resulting behaviors. A batch typically contains 20 runs, each of which starts with the same default organism capable only of self-replication. All other behaviors must enter the genome through mutations. Since each run within a batch starts with a different random number seed, the populations take different evolutionary paths.

Cooperative communication

The first study addresses the evolution of cooperative communication algorithms. Sensor networks often employ complex distributed operations such as multicasting, gathering sensed data, and detecting and

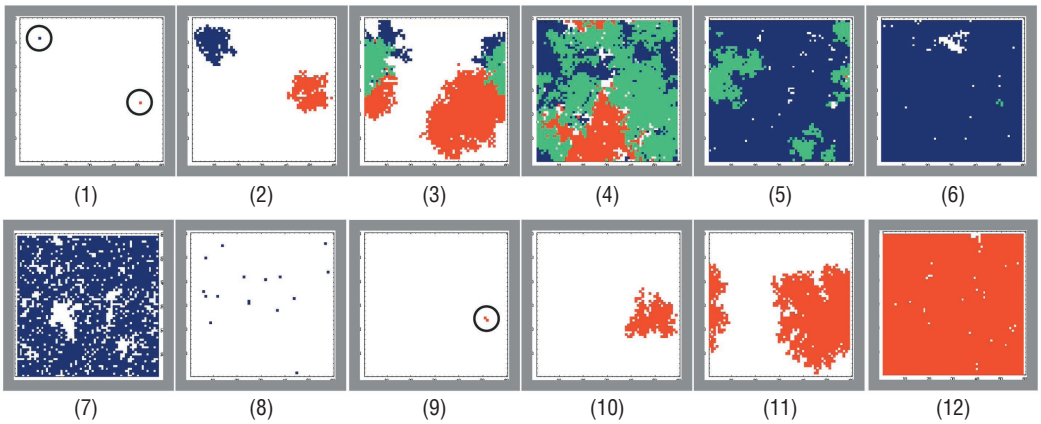

Figure 3. Snapshots of an Avida population in a 60×60 grid. The snapshots demonstrate distribution of the largest sensed value (blue) and, when that value resets, the next-largest value (red). Organisms sending both messages appear in green.

responding to events of interest. Unfortunately, many traditional algorithms for solving these problems are brittle when deployed in dynamic environments, and improvements are limited to the methods considered by human designers.

On the other hand, digital evolution provides a means to explore a larger solution space, potentially discovering algorithms more likely to remain effective even under extremely adverse conditions. For digital organisms to thrive in highly dynamic environments such as Avida, where organisms continually replace one another, they must evolve resilient solutions.

Consider the evolution of a particular distributed problem-solving task. We assign each cell in the environment a random 32-bit identifier, which a resident organism can sense using the GET-ID instruction. The organisms must determine the largest sensed value and distribute it throughout the population. Performing this operation could provide a basis for a leader-election algorithm, or a wireless sensor network could use it to obtain and distribute the maximum sensed value.

We designed a set of tasks to reward Avidians with more virtual CPU time for exhibiting cooperative behaviors and to penalize them with less when they did not. Over time, the population evolved to identify the largest value. To further assess the robustness of their solution to the problem, once the largest value had been discovered, we removed it from the population. This forced the population to continually search for the maximum value.

Figure 3 shows snapshots of a population that evolved these behaviors. The snapshots show the spread of two different values. Each snapshot identifies which organisms send the largest cell ID (blue), which send the second-largest cell ID (red), and which send both (green). By frame 6, nearly all organisms are sending messages that carry the largest cell ID; a few organisms near the cell with the second-largest ID send both. We reset the largest cell ID just prior to frame 7. As shown, the transmission of that cell ID dies out quickly. The population, however, recovers and proliferates messages that carry the new largest cell ID.

Figure 4 shows the dominant genome from the population exhibiting this behavior. This particular genome comprises 85 instructions, of which 11 are responsible for the desired behavior, 22 implement the organism's replication cycle, one instruction is shared, and 51 instructions—or 60 percent of the genome—are neutral mutations that do not affect the organism's phenotype. Genomes of living organisms, including humans, also contain large percentages of "junk DNA," the role of which researchers do not completely understand, but which might include serving as building blocks for new functionality.

Interestingly, this particular genome has a spin-wait near the top of the highlighted code segment, which effectively makes the organism's replication dependent upon receiving a message that carries a cell ID larger than its own. Organisms with this genome have evolved to the point where they depend upon other organisms' behavior for their survival: If an organism does not receive a message that has a data field larger than its own cell ID, it will not reproduce.

Energy management

Mobile devices with limited battery resources must conserve energy. For example, communication traffic flowing through an ad hoc wireless network directly affects the energy consumption at individual nodes, and excessive or disproportionate energy consumption can lead to node failure and possibly network partitioning. Determining the optimal energy management strategy in such situations involves many factors—such as dynamic flows, physical topology, movement constraints, security concerns, and energy consumption—and a multitude of possible scenarios. This part of our research investigates whether digital evolution can yield energy-efficient algo-

rithms and protocols that perform well under dynamic and adverse conditions.

Our early studies focus on the evolution of sleep behavior in digital organisms.[7] We subjected populations of Avidians to an environment with a slowly diminishing resource and recorded their ability to adapt to the changing environment using sleep instructions. These instructions let organisms enter a low-energy state that lasts for multiple CPU cycles. Avidians were rewarded for performing simple computational tasks—logic operations—but only when this resource, the digital equivalent of sunlight, was available. The resource was available a percentage of each 256-time-step Avidian day, but that percentage declined with each passing year of 500 Avidian days.

We observed that Avidians adapted to use sleep instructions effectively, despite the risk that a sleeping organism might be replaced before it reproduced. Examination of genomes showed that some populations evolved a behavior we anticipated, where organisms would sleep for short intervals and periodically wake to check for the resource. However, a majority of the populations evolved an unexpected behavior, the equivalent of a biological alarm clock, that adjusted the length of the gestation cycle to synchronize with the resource's availability. Moreover, experiments revealed that organisms evolved to start sleeping just before the resource went away and—just prior to the return of the resource—to awaken and begin preparing data to be used in tasks.

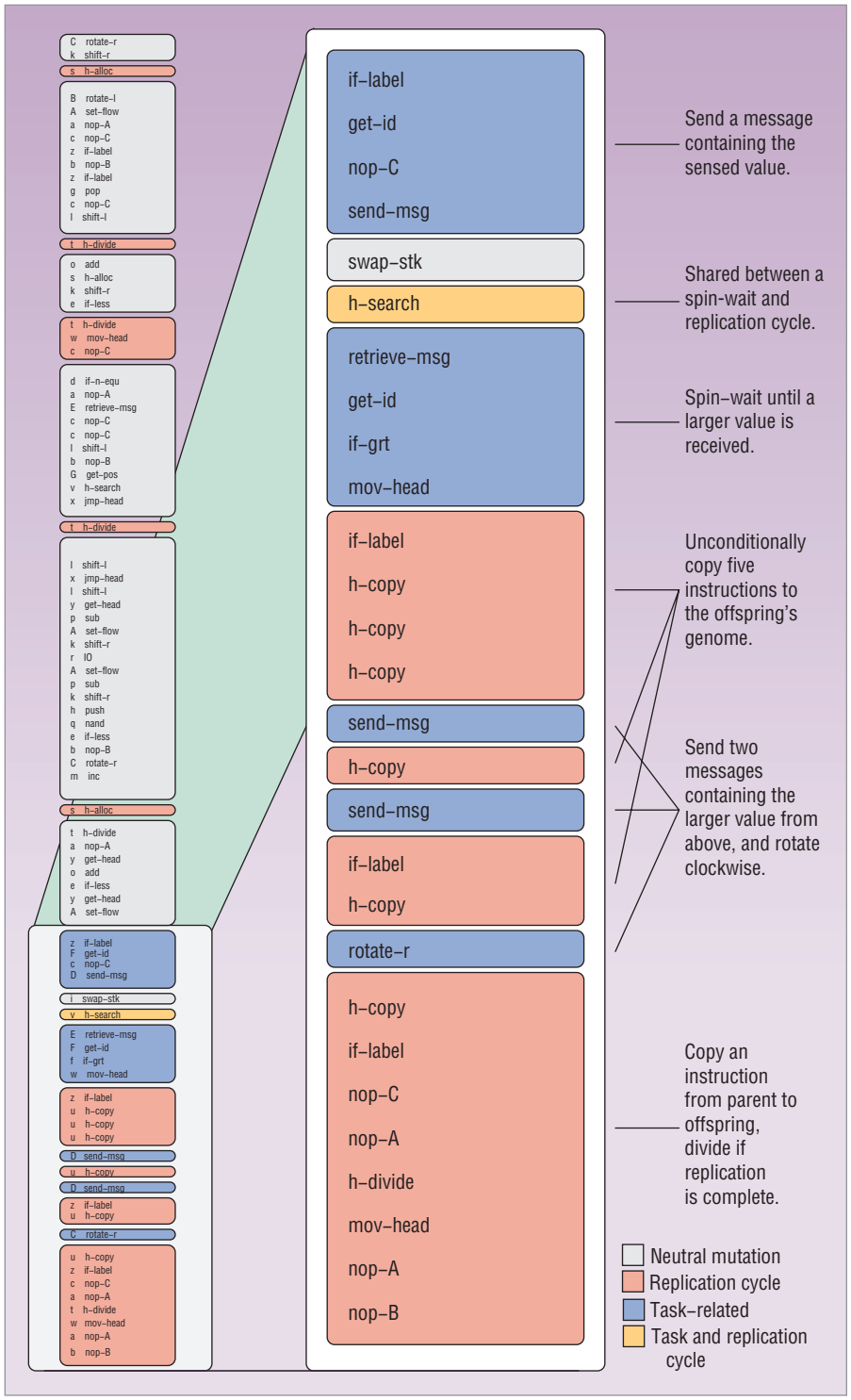

Figure 4. Dominant genome in a population that evolved to identify and distribute the largest cell ID. The full genome appears on the left. The expanded section of the genome shows how evolution co-opted the organism's replication cycle and inserted logic to help perform the task.

This "early to bed, early to rise" behavior lets organisms finish tasks early during periods of resource availability, thereby increasing the probability of receiving a reward. It also helps avoid situations in which an organism starts working on a task but completes it just after the resource disappears, when there is no reward. Figure 5 shows a sample population that evolved this behavior, recorded in snapshots of a 60×60 grid during a single Avidian day.

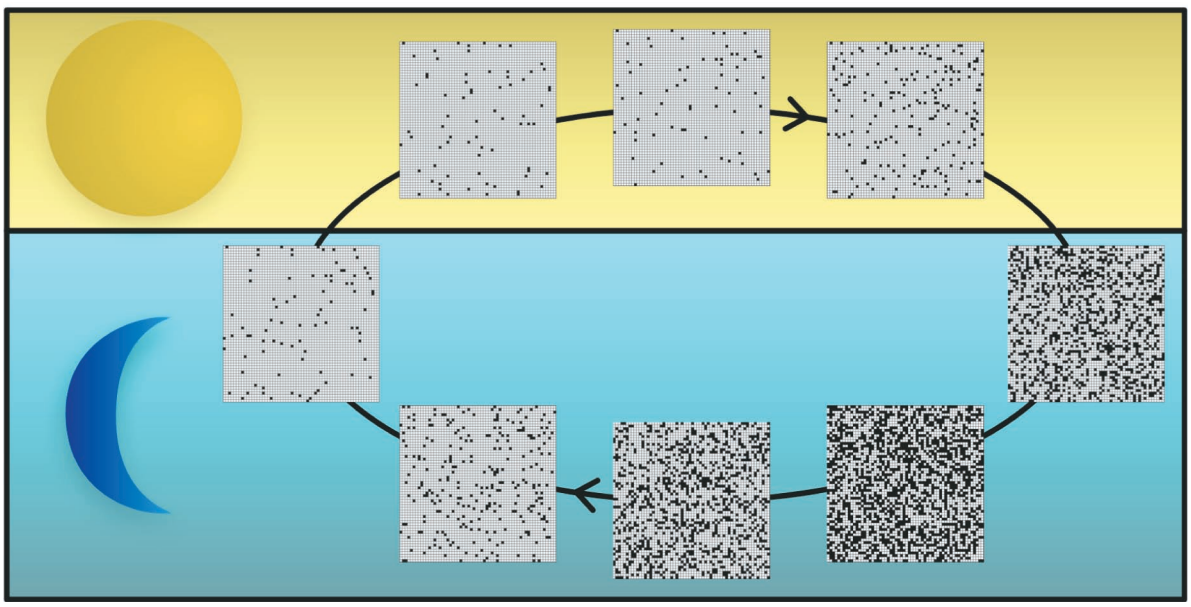

Figure 5. Representations of a population's response to the resource availability during an Avidian day. Black squares represent sleeping organisms, white squares represent awake ones. Snapshots in the figure's top half show the population when the resource is available, and they are rewarded for completing tasks. Snapshots in the figure's bottom half show the population when the resource is not available, and task completion goes unrewarded.

At this point in the run, the resource is available about 44 percent of the day. The black squares depict sleeping organisms, the white squares awake ones. The three snapshots at the top part of the figure depict the population's state when the resource is available, while the five snapshots on the bottom part depict the population's state when the resource is unavailable. This adaptive behavior arose in 37 out of 50 runs.

Our ongoing studies address conservation of energy balanced against other activities, such as detecting and reporting events of interest. We plan to test the most promising evolved solutions on sensor network simulators and, eventually, deploy them on physical devices and compare them to hand-built solutions.

Evolving behavioral models

We also use digital organisms to assist in constructing models of software behavior, including adding new functionality to an existing system. Software developers often use model-driven development (MDD)[8] to construct graphical models of desired structure and behavior, automatically transform the models into more formal specifications, and eventually generate the corresponding code.

Currently, many developers use the Unified Modeling Language to model systems. Despite MDD's many advantages, however, the construction of a UML behavioral model—which comprises a set of state diagrams for interacting objects—can be error-prone and difficult to automate, especially when extending an existing model to include new functionality. Digital evolution provides a means to generate possible solutions automatically.

Our approach, depicted in Figure 6, treats each Avidian as a generator of state diagrams: When the organism executes, it constructs an in-memory representation of one or more state diagrams. To implement this method, we extended Avida and integrated it with existing software engineering tools. First, we provide each organism with information about class diagram elements and, optionally, any existing state diagrams of the system. We call this information *instinctual knowledge*.

When replication creates a new organism, it is provided with a file containing its instinctual knowledge. For every class in the class diagram, the file contains an optional existing state diagram and lists of elements—such as triggers, guards, actions, and states. We also enhanced the Avida instruction set with instructions that let an organism use its instinctual knowledge to create additional transitions in one or more state diagrams.

For a given problem, the developer defines a collection of tasks that reward organisms for generating state diagrams that support scenarios, satisfy formally specified properties, and optimize software engineering metrics, such as minimizing the number of transitions in a diagram. To enable Avida to assess the completion of such tasks, we integrated it with a UML formalization framework, Hydra,[9] and the Spin model checker.[10] Hydra translates generated state diagrams into a representation in the Promela specification language, which Spin verifies against properties.

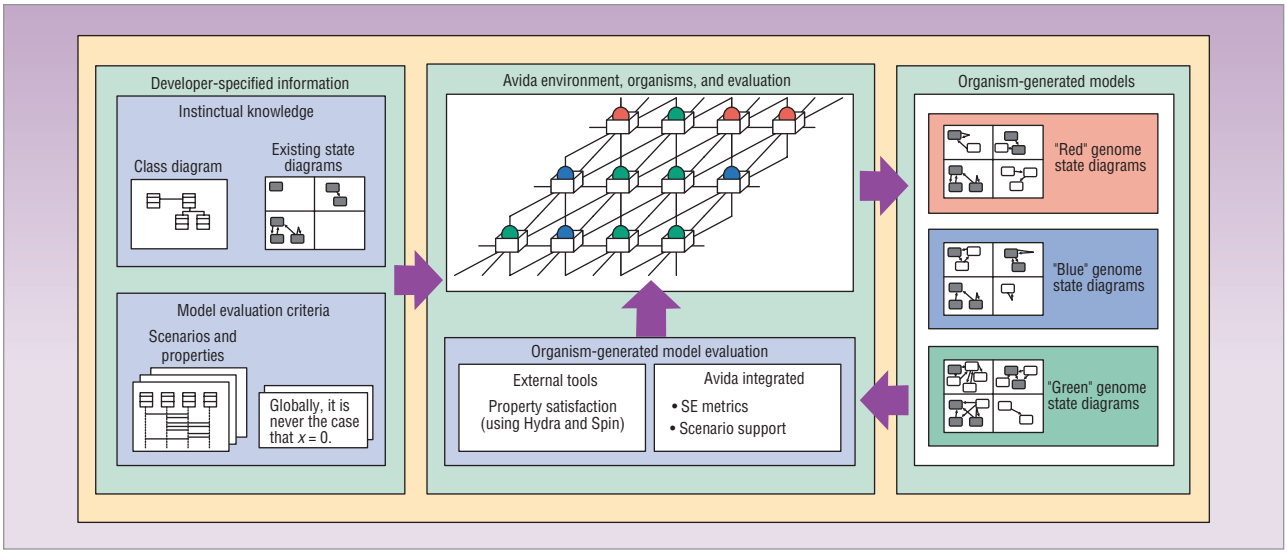

Figure 6. Using Avida to develop software state diagrams. Individual organisms are provided with instinctual knowledge of existing software and evolve to produce state diagrams that meet developer-specified requirements.

As in other Avida applications, a population starts with a single organism capable only of replication. As the organism and its descendants replicate, random mutations produce different genomes. Organisms that generate state diagrams exhibiting desired characteristics receive more CPU cycles and thus replicate faster.

Effectively, an Avida population is subject to a natural-selection pressure that rewards organisms for generating state diagrams that support key scenarios and satisfy critical properties. If an organism generates state diagrams that support all key scenarios and satisfy all properties, it has successfully and automatically generated a behavioral model for the system. We refer to the state diagrams that meet these requirements as *compliant* state diagrams. At this point, the experiment succeeds and we can halt it, or we might allow it to proceed to find other sets of compliant state diagrams. We used this approach to generate state diagrams describing new mobility behavior in a robot.[11] Researchers and developers can apply this technology to other domains exhibiting complex requirements.

FUTURE DIRECTIONS

Although evolutionary computation is a well-established computing subfield, we are just beginning to understand how to harness the evolution of self-replicating digital organisms. Several major lines of research offer opportunities for those interested in this area of study.

The first involves different architectures and instruction sets. Avida is an extensible platform, and various von Neumann CPU architectures have been implemented and used in past studies. Within the current Avida environment, we are investigating instruction sets with better support for flow control, function invocation, and context switching. However, fundamentally different computation models, such as data flow machines or even models based on processors found in natural systems, such as gene regulatory networks, might lead to the evolution of complex and adaptive behaviors.

We also plan to expand our work on evolving digital organisms to construct models of software and other aspects of computing systems. Integrating Avida with tools for automated software engineering helps address the increasing need for high-assurance, robust software that can tolerate adverse physical conditions and flaws in hardware fabrication. Moreover, Avidians can evolve to help design other structures—such as network topologies—important to distributed computing.

Mobility presents another major area of future study. Members of our group have recently modified Avida to let organisms move among cells, and we have started developing a continuous-space Avida environment in which the laws of physics govern movement and communication. We are particularly interested in the evolution of cooperative mobility control. Coordination of movements is critical to behaviors such as flocking, avoiding obstacles, and eluding enemies. Moreover, recent studies with mobile sensors have shown that it's possible to exploit mobility to provide certain benefits to network performance, energy conservation, and communication security. A fundamental question is whether digital evolution might find behaviors that enable a collection of mobile robots to adapt to, and perhaps exploit, current conditions in ways not otherwise apparent to human designers.

A related area of study involves integrating biomimetics and digital evolution. Evolution has produced complex behaviors in natural systems, which might provide an effective starting point for evolving control software for

Related Research

Research into harnessing evolution extends into both the design and behavior of virtual and embodied agents and machines. Evolution has been harnessed to create more realistic videogames, MEMS chips that operate under extreme conditions, and swarm behavior in robots. Research papers in this area can be found in journals and conferences sponsored by the IEEE Computational Intelligence Society, the ACM Special Interest Group for Genetic and Evolutionary Computation, and the International Society of Artificial Life, among others.

Table A displays a small sampling of the groups conducting research in this field. While these groups use widely varying underlying substrates, they all share the concept of harnessing evolution and using it to solve problems.

Table A. Sampling of groups applying evolutionary computing to systems design.

Laboratory/Group	Institution/Organization	Keywords
Neural Networks Research Group	University of Texas at Austin	Neuroevolution, self-organization, robotics, evolutionary computation
Dynamical and Evolutionary Machine Organization Laboratory	Brandeis University	Coevolution, evolutionary robotics, neuroevolution
Cornell Computational Synthesis Laboratory	Cornell University	Evolutionary robotics, modular robotics, rapid prototyping
IRIDIA Laboratory	Free University of Brussels	Swarm intelligence, swarm-bots, self-organizing systems, biological networks
Laboratory of Intelligent Systems	École Polytechnique Fédérale de Lausanne	Flying robots, artificial evolution, social systems
Adaptive Control and Evolvable Systems Group	US National Aeronautics and Space Administration	Automated design, system optimization
Digital Biology Interest Group	University College London	Evolutionary computation, bio-inspired computing, developmental systems
Evolutionary Computation Laboratory	University of Central Florida	Neuroevolution, coevolution, autonomous agents
Bionics and Evolutiontechnique Department	Technische Universität Berlin	Bio-inspired machines, bionics
Adaptive Computation Group	University of New Mexico	Artificial immune systems, genetic algorithms, biological modeling
Evolutionary and Adaptive Systems Group	University of Sussex	Artificial life, evolutionary computation, adaptive systems

robots. For example, some animal species exhibit fission-fusion relationships in which individuals join together for some tasks, such as guarding a den or attacking prey, but act independently at most other times. We can hand-code such behaviors in an Avida organism and use it to seed the evolutionary process. Evolution in Avida would likely modify the behaviors to account for differences between robots and animals, including both enhanced capabilities such as availability of radio communication and limitations such as physical agility.

Finally, we can explore the joint evolution of the system's morphology, or physical structure, and its control software. Several researchers use evolutionary computation to help design integrated software and hardware for robots.[12] After all, organisms' bodies and brains evolve together in nature. Indeed, some would argue that intelligent behavior can evolve only when the system's decision-making part is coupled with a physical body that has sensors and actuators. Others claim that the sense-and-respond functionality can be abstracted from the physical world (into software sensors and actuators, for example) and still lead to evolution of intelligent behavior. Using digital evolution, we have begun studies to help answer this question.

Our preliminary studies using Avida to evolve interesting behaviors show promise and open doors to several areas of future research. In addition, the "Related Research" sidebar profiles several other research groups that apply various forms of evolutionary computation to systems design. This problem domain appears to offer a fertile research area with potentially important implications, given the increasing complexity of computing systems. We hope this research community will continue to grow.

Further information on our research can be found at www.cse.msu.edu/thinktank. For papers on other digital evolution applications, and Avida downloads and accompanying documentation, see http://devolab.cse.msu.edu.

Acknowledgments

We gratefully acknowledge the contributions of the faculty and students in the Digital Evolution Laboratory at Michigan State University. This research was supported in part by the Michigan State University Quality Fund, the US Department of the Navy, Office of Naval Research under grant no. N00014-01-1-0744, the DARPA Fundamental Laws of Biology program, and National Science Foundation grants ITR-0313142, CCF-0523449, CCF-0541131, and CCF-0750787.

References

1. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41-50.
2. P.K. McKinley et al., "Composing Adaptive Software," *Computer*, July 2004, pp. 56-64.
3. D. Floreano, P. Husbands, and S. Nolfi, "Evolutionary Robotics," *Handbook of Robotics*, Springer-Verlag, 2008.
4. C. Adami, *Introduction to Artificial Life*, Springer-Verlag, 1998.
5. D.C. Dennett, "The New Replicators," *The Encyclopedia of Evolution*, M. Pagel, ed., vol. 1, Oxford Univ. Press, 2002, pp. E83-E92.
6. R.E. Lenski et al., "The Evolutionary Origin of Complex Features," *Nature*, vol. 423, 2003, pp. 139-144.
7. B. Beckmann, P.K. McKinley, and C.A. Ofria, "Evolution of Adaptive Sleep Response in Digital Organisms," *Proc. 9th European Conf. Artificial Life*, Springer, 2007, pp. 233-242.
8. D.C. Schmidt, "Model-Driven Engineering," *Computer*, Feb. 2006, pp. 25-31.
9. W.E. McUmber and B.H.C. Cheng, "A General Framework for Formalizing UML with Formal Languages," *Proc. IEEE Int'l Conf. Software Eng.* (ICSE 01), IEEE Press, May 2001, pp. 433-442.
10. G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*, Addison-Wesley, 2004.
11. H.J. Goldsby et al., *Automatic Generation of UML Behavioral Models through Digital Evolution*, tech. report MSU-CSE-07-194, Dept. Computer Science and Eng., Michigan State University, East Lansing, Mich., 2007.
12. H. Lipson, "Evolutionary Robotics and Open-Ended Design Automation," *Biomimetics*, B. Cohen, ed., CRC Press, 2005, pp. 129-155.

Philip McKinley *is a professor in the Department of Computer Science and Engineering at Michigan State University, East Lansing, Michigan. His research interests include self-adaptive software, autonomic computing, and digital evolution. McKinley received a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the ACM. Contact him at mckinley@cse.msu.edu.*

Betty H.C. Cheng *is a professor in the Department of Computer Science and Engineering at Michigan State University. Her research interests include model-driven engineering, formal and automated analysis of high-assurance systems, and adaptive and autonomic systems. Cheng received a PhD in computer science from the University of Illinois at Urbana-Champaign. She is a senior member of the IEEE. Contact her at chengb@cse.msu.edu.*

Charles Ofria *is an assistant professor in the Department of Computer Science and Engineering and the Ecology, Evolutionary Biology, and Behavior Program at Michigan State University. His research interests include digital evolution, biocomplexity, and bioinformatics. Ofria received a PhD in computation and neural systems from the California Institute of Technology. Contact him at ofria@cse.msu.edu.*

David Knoester *is a doctoral student in the Department of Computer Science and Engineering at Michigan State University. His research interests include digital and biological evolution, self-organizing systems, and distributed computing systems. Knoester received an MS in computer science from Michigan State University. He is a member of the IEEE and the ACM. Contact him at dk@cse.msu.edu.*

Benjamin Beckmann *is a doctoral student in the Department of Computer Science and Engineering at Michigan State University. His research interests include sensor networks, autonomic computing, and evolutionary robotics. Beckmann received an MS in computer science from Western Michigan University. He is a member of the IEEE and the ACM. Contact him at beckma24@msu.edu.*

Heather Goldsby *is a doctoral student in the Department of Computer Science and Engineering at Michigan State University. Her research interests include model-driven development of high-assurance systems, dynamically adaptive systems, and using digital evolution to support software development. Goldsby received an MS in computer science from Michigan State University. She is a member of the IEEE and the ACM. Contact her at hjg@cse.msu.edu.*