Problem Decomposition Using Indirect Reciprocity in Evolved Populations

Heather J. Goldsby, Sherri Goings, Jeff Clune, Charles Ofria
Department of Computer Science and Engineering
Michigan State University
3115 Engineering Building
East Lansing, Michigan 48824 USA
{hjg, goingssh, jclune, ofria}@msu.edu

ABSTRACT

Evolutionary *problem decomposition* techniques divide a complex problem into simpler subproblems, evolve individuals to produce subcomponents that solve the subproblems, and then assemble the subcomponents to produce an overall solution. Ideally, these techniques would automatically decompose the problem and dynamically assemble the subcomponents to form the solution. However, although significant progress in automated problem decomposition has been made, most techniques explicitly assemble the complete solution as part of the fitness function. In this paper, we propose a digital-evolution technique that lays the groundwork for enabling individuals within the population to dynamically decompose a problem and assemble a solution. Specifically, our approach evolves specialists that produce some subcomponents of a problem, cooperate with others to receive different subcomponents, and then assemble the subcomponents to produce an overall solution. We first establish that this technique is able to evolve specialists that cooperate. We then demonstrate that it is more effective to use a generalist strategy, wherein organisms solve the entire problem themselves, on simple problems, but that a specialist strategy is better on complex problems. Finally, we show that our technique automatically selects a generalist or specialist strategy based on the complexity of the problem.

Categories and Subject Descriptors

F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Self-modifying machines*

General Terms

Experimentation.

Keywords

Digital evolution. Altruism. Binary String Cover Problem. Evolution of Cooperation.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

1. INTRODUCTION

Traditionally, evolutionary algorithms (EAs) contain a single population in which each individual encodes a complete solution to a problem. EAs work well for problems with a search space that can be reasonably well-explored, given the computational resources available to the population. However, as the complexity of the problems addressed using EAs increases, the size of the search space and the amount of time required to search for a satisfactory solution grows rapidly. *Problem decomposition* has been proposed to address this limitation [15]. Problem-decomposition EAs decompose a problem into simpler subproblems, individuals produce *subcomponents* (where each subcomponent is a solution to a subproblem), and then the subcomponents are combined to obtain a solution to the overall problem [15]. One of the overarching challenges of problem-decomposition EAs is to have the population automatically and dynamically both decompose the problem (if necessary) and assemble a complete solution.

Although significant progress has been made in problem decomposition [3, 6, 10, 15, 17, 19], most techniques evolve specialists that produce subcomponents and explicitly recombine the subcomponents as part of the fitness function to produce an overall solution [15, 19]. A notable exception is the approach taken by some learning classifier systems (LCSs), which evolve specialists that produce and then dynamically assemble the subcomponents to form a solution [10, 17]. For example, this LCS technique been used to evolve the behavior of robots in real-world applications [4]. However, LCSs are designed for rule-based problems and are thus restricted in their application. To enable problem-decomposition EAs to solve complex, non-rule-based problems, we must explore other techniques for dynamically decomposing a problem and assembling a solution.

In this paper, we propose a technique for problem decomposition that lays the groundwork for evolving specialists that automatically decompose a problem, produce subcomponents, and then dynamically assemble the subcomponents to form a complete solution. Specifically, our technique currently evolves specialists that produce subcomponents and cooperate with other neighboring specialists to collect all of the subcomponents necessary to form a complete solution. Because an individual organism collects all of the subcomponents required for the complete solution, this approach has the potential to be used to address problems that require more complex assembly procedures, which can also be evolved.

We implement our approach in AVIDA [14], a digital-evolution platform previously used to study the origin of complex features [9], the evolutionary design of modularity [12], evolutionary robustness [8, 18], and the evolution of altruism [1, 5]. Within an AVIDA experiment, a population of self-replicating computer programs exists in a user-defined computational environment and is subject to mutations and natural selection. These "digital organisms" compete for limited resources and are subjected to mutations.

For this study, we provide instructions and infrastructure to allow organisms to cooperate in order to obtain subcomponents produced by neighboring organisms using *indirect reciprocity* [11]. Whereas *direct reciprocity* involves the exchange of altruistic donations during repeated interactions between the same two individuals, indirect reciprocity involves the exchange of donations for reputation, where future donations are directed toward individuals with a high reputation [11, 16]. In this study, an organism's reputation is an accurate reflection of its behavior that is automatically updated. Specifically, an organism can donate a subcomponent to a neighboring organism. As a result of this donation, the donor organism's reputation automatically increases. This improved reputation, in turn, increases the probability that the donor organism will subsequently receive donations of subcomponents from others. Thus, an organism can assemble a complete solution by combining the subcomponents it has produced itself with subcomponents that it has received from others.

We demonstrate the potential for this approach using a variant of the binary string cover problem presented in [15]. Whereas the original binary string cover problem considers a *set of individuals* to be a solution, we explicitly require *individuals* to cooperate to assemble a complete solution. In this case, assembling a solution simply requires an organism to collect all of the subcomponents. First, to test whether this approach can evolve cooperation, we force individuals to be *specialists* that produce only one subcomponent and cooperate to solve the complete problem. Next we compare the performance of a population of *generalists* that each attempt to solve the entire problem to the performance of a population of specialists. On simpler problems, the population of generalists outcompete the population of specialists. However, as the complexity of the problem increases, the population of specialists outperforms the population of generalists. Lastly, we demonstrate that when allowed to be either generalists or specialists, the population composition automatically changes to solve the problem most efficiently. Specifically, we enable both generalists and specialists to coexist within a population. We run experiments of varying complexity and show that the population composition changes from consisting of primarily generalists to primarily specialists as the problem complexity increases.

2. RELATED WORK

Numerous techniques have been proposed to automate the evolution of problem decomposition [3, 6, 10, 15, 17, 19]. In general, these techniques evolve specialists that produce subcomponents. The overall solution is considered to be a group of specialists comprising either one representative member of each species (e.g., [6, 15, 19]) or an EA-selected team (e.g., [3, 10, 17]).

Several approaches assemble the subcomponents as part of the fitness function [15, 19]. Specifically, the cooperative coevolution architecture [15] evolves two or more species in completely isolated populations. Cooperation between these species occurs at the time of fitness evaluation, when individuals from one species are evaluated with representatives from each of the other species. The nature of these collaborations is determined by the user, as is the choice of representatives from each species, but the number of species is determined by the algorithm automatically. Additionally, Yong and Miikkulainen [19] apply the cooperative coevolution architecture to evolve coordinated predator behavior in a prey-capture task. Three Enforced SubPopulations [7] are created, each of which evolves an artificial neural network that represents the behavior for one predator [19]. During the fitness evaluation, a representative neural network from each subpopulation is placed in a simulated environment and the reward for captured prey is split equally among all representatives present. Unfortunately, these approaches are only applicable in problems where the general form of between-species collaborations is known *a priori*.

Other approaches enable the EA itself to dynamically assemble the subcomponents. These approaches are typically based on a market economy. For example, some learning classifier systems form teams of individuals that bid for their subcomponents to be used as part of the solution to the classification problem [10, 17]. Additionally, Cornforth and Kirley [3] propose a non-rule based approach to problem decomposition using a market-based model, where agents group together in a hierarchical fashion to form complex problem solutions. One limitation of this approach is that individuals do not dynamically evolve subcomponents of the problem, but rather innately represent preformed subcomponents.

3. AVIDA

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 that executes those instructions. The standard AVIDA instruction set is Turing complete and is designed so that random mutations will always yield a syntactically correct program, albeit one that may not perform any meaningful computation [13]. Within their virtual environment, organisms can produce and consume resources, aid neighboring organisms through altruistic donations, and sense or change various properties of the environment.

An AVIDA population comprises a number of *cells*, where a cell is a compartment in which an organism can live. Each cell can contain at most one organism, and the size of an AVIDA population is bounded by the number of cells in the environment. Cells are connected to each other via a configurable topology, generally a grid or torus, that defines the neighborhood of each cell. An organism's *neighbors* are the organisms that live in the cells adjoining its own cell. Additionally, each organism has a *facing*, which is one connection to a neighboring cell. Organisms are able to change their facing by executing rotate instructions.

Organisms are *self-replicating* meaning that the genome itself must contain the instructions to create an offspring. When an organism replicates, a cell to contain the offspring is selected from the environment at random, and any previous inhabitant of the target cell is replaced (killed and overwritten) by the offspring. Each population starts with one or more organisms that are capable only of replication,

and different genomes are produced through random mutations introduced during replication. Mutation types include: replacing the instruction with a different one, inserting an additional, random instruction into the offspring's genome, and removing an instruction from the offspring's genome.

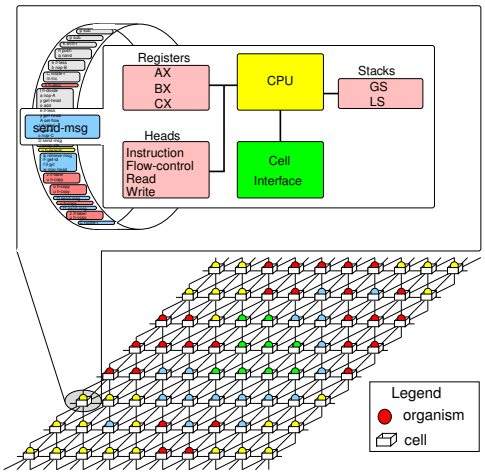


Figure 1: The AVIDA platform

Organisms can perform *tasks* that allow them to metabolize resources from their environment. Metabolizing beneficial resources increases an organism's *merit*, which determines the rate its virtual CPU will execute instructions relative to the other organisms in the population. For example, an organism with a merit of 2 will, on average, execute twice as many instructions as an organism with a merit of 1. Since digital organisms are self-replicating and compete for space, a higher merit (all else being equal) results in an organism that replicates more rapidly, spreading throughout and eventually dominating the population.

The amount of merit that an organism gains for completing a task depends on the type and abundance of resources associated with the task. These resources may be *unlimited*, such that they are always available and the merit increase gained for completing them is effectively a user-defined constant. Or the resources may be *limited*, such that the merit increase gained is dependent on the amount of the associated resource that is currently available. A small amount of each limited resource continually flows into the environment to replenish the resource stores. When many organisms execute a task associated with a limited resource, the amount of resource available decreases until additional organisms would receive more merit by completing a task associated with a different resource. Cooper and Ofria have shown that using limited resources in AVIDA leads to greater diversity and that with them multiple species can stably coexist in an asexual population [2]. In this paper, we use tasks associated with both limited and unlimited resources.

4. EXPERIMENTAL SETUP

In this section, we provide an overview of the binary string production problem that we use to illustrate our technique, discuss the modifications to AVIDA that were necessary to allow specialists to cooperate, and provide the standard configurations used for our experiments.

4.1 Binary String Production Problem

The original binary string cover problem, which we are using a variation of, considers each individual in a population to represent a binary string x. The objective of each individual is to match as strongly as possible a set V of N binary vectors called *strings*, where each string v_i in V is of length l. Let x be a match value, where b_i is the number of bits exactly matched (value and placement) between x and v_i. Formally, x is defined as follows:

- If $b_i < \frac{l}{2}$, then $S(x, v_i) = 0$.
- If $b_i \geq \frac{l}{2}$, then $S(x, v_i) = (\frac{2b_i}{l} - 1)^2$.

Unless the strings in set V are identical, no single value of x will be able to perfectly optimize all objectives, and thus trade-offs must occur. The EA's solution to the binary string cover problem is typically considered to be a set of individuals in the final population [6, 15]. If individuals within the population specialize, then they are able to produce a better overall solution. Goings and Ofria have previously demonstrated the ability of AVIDA to develop specialists that cover multiple niches in this problem [6].

Whereas the original binary string cover problem statically composes subcomponents by considering a set of individuals (one from each species) to represent the solution, this variation increases the difficulty of the original problem by requiring each individual to produce or acquire all of the strings and assemble the complete solution. We refer to this variation as the *binary string production problem*. Thus, the objective of the individual is, given a set V of binary strings, to produce multiple exact copies of V. We refer to a perfect copy of V as a *complete set*.

For our experiments, we use several variants of the binary string production problem, where each variant differs in complexity. Specifically, we define 4-bit, 8-bit, 16-bit, and 24-bit variants. Each variant has two strings that constitute V: the first string is all zeros and the second string is all ones. For example, the 4-bit variant uses strings 0000 and 1111 and the complete set is {0000, 1111}.

4.2 Avida Extensions

To enable organisms to specialize and cooperate to produce complete solutions, we extended the organism infrastructure, defined instructions to allow organisms to produce complete sets, and defined tasks that described the desired outputs. At a high level, an organism creates a string in its buffer and then produces copies of the string. The copies can be donated to increase its reputation or used as parts of a complete set. Tasks are associated with the quality of strings produced, quantity of strings produced, and also the quantity of complete sets produced.

4.2.1 Organism Infrastructure

Figure 2 (a) depicts the four key elements of an AVIDA organism that are used to produce solutions to the binary string production problem. First, an organism has a *buffer* in which it is able to construct strings. The buffer is initially empty and is exactly the length of a string. A pointer into the buffer determines where insertion will occur. The pointer starts at the first element, and moves one position each time a bit is inserted into the buffer. After the last element has been written, the pointer returns to the first element and future insertions overwrite existing bits. For example, Figure 2 (a) depicts a buffer of length 4 (for the 4-bit variant of the string production problem). Second, an

organism has two counters that track the number of copies of each string that the organism has at its disposal, where a copy could be either produced by the organism or received as a donation. For example, Figure 2 (a) depicts the two counters (c_{0000} and c_{1111}). The number of copies of a string is limited to a user-defined amount. For these experiments, the limit is 10 copies per string. A copy can be used as part of a complete set or as a donation. Third, an organism has a *tag* that identifies which vector it is best at producing. An organism initially inherits the tag of its parent. As the organism replicates, its tag and the tag of its offspring are updated to reflect the vector it is best at producing. Lastly, an organism has a reputation that is automatically updated as a result of the organism's cooperative actions. An organism's reputation is determined using a variant of a standing strategy [16] as follows [1]:

- **reputation = 0**: An organism has neither donated a string nor received a string. All organisms are born in this state.
- **reputation = -1**: An organism has not donated a string, but has received a string.
- **reputation = 1** An organism has donated a string and may or may not have received a string.

4.2.2 Instructions

Table 1 depicts the instructions we added to the standard AVIDA instruction set [13]. Instructions insert-0 and insert-1 insert a 0 or a 1 into the organism's buffer. Figure 2 (b) depicts an organism after it executed insert-0. Instruction prod-string reads an organism's buffer. If the string in the buffer perfectly matches one of the vectors in V, the organism's counter for this vector is incremented by 1; otherwise, the instruction has no effect. For example, Figure 2 (c) depicts the buffer of an organism that has constructed string 0000 and then executed prod-string. The counter for the string 0000, c_{0000} is incremented. The buffer remains the same after the prod-string instruction is executed. Thus, an organism may consecutively execute the instruction several times to create multiple copies of the same string.

The remainder of the new instructions allow organisms to cooperate through indirect reciprocity. To donate a string, an organism executes the instruction donate-string, which decrements the organism's counter for the string and increments the corresponding counter of the neighbor. An organism's tag determines which string it donates. For example, if an organism's tag is 0000, then the organism donates string 0000. Donations between organisms of the same tag would not assist organisms in constructing a complete set and thus are not allowed. Additionally, if an organism does not have any copies of the string it donates (e.g., $c_{0000} = 0$), then the donation does not occur and the counters and reputations of the organism and its neighbor remain unchanged.

Organisms are able to sense their own reputation and the reputation of their facing neighbor using the get-reputation and get-neighbors-reputation instructions, respectively. When these instructions are executed, the reputation value is placed in one of the organism's registers. Lastly, the rotate-to-rep-tag instruction rotates an organism to face the neighbor with the highest reputation of the opposite tag. If the organism does not have a neighbor of the opposite tag, then its facing remains unchanged. If multiple neighboring organisms of equally high reputation all have the opposite tag, then the organism is rotated to face a random organism from this set.

[1]We also experimented with reputation strategies in which an organism's reputation increased with each subsequent donation. These strategies did not qualitatively change our results.

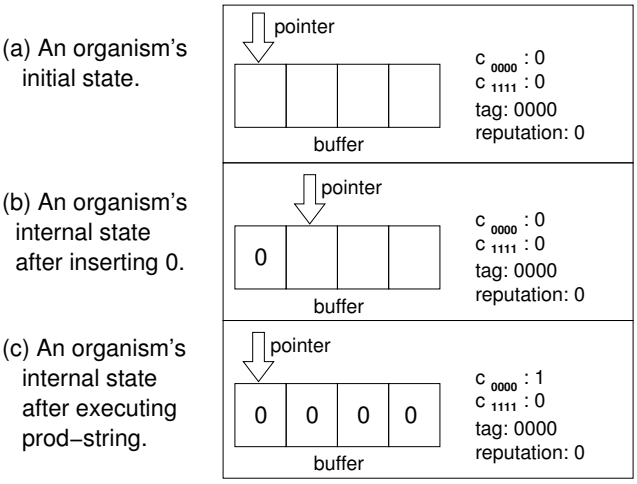


Figure 2: The infrastructure that allows Avida organisms to produce strings, donate strings, and cooperate using indirect reciprocity.

Table 1: Additional Avida instructions that allow organisms to produce strings, donate strings, and selectively cooperate with neighbors on the basis of reputation.

Instruction	Description
insert-0	Insert 0 into the buffer
insert-1	Insert 1 into the buffer
prod-string	If the string in the buffer is a vector v_i, increment the counter of v_i
donate-string	Donate a copy of a string to the facing neighbor
get-reputation	Get the organism's reputation
get-neighbors-reputation	Get the neighbor's reputation
rotate-to-rep-tag	Rotate to the neighbor with a different tag and the highest reputation

4.2.3 Tasks

We defined two types of tasks associated with resources that reward organisms for the binary string production problem. The first task, MatchString, is associated with a limited resource and is used to ensure that the population maintains the ability to produce both strings. The performance of an organism on MatchString is proportional to the quality of the string in its buffer when the organism replicates. A MatchString task is defined for both strings in V (e.g., $\text{MatchString}_{0000}$ and $\text{MatchString}_{1111}$) and the value of the task is defined using the same formula as the original bit string cover problem. Additionally, if the organism has produced the string that perfectly matches the vector itself, then it receives a score of 1. The formula is:

- If $b_i < \frac{l}{2}$, then $MatchString(org, v_i) = 0$.
- If $b_i \geq \frac{l}{2}$, then $MatchString(org, v_i) = (\frac{2b_i}{l} - 1)^2$.
- If $c_{v_i} \geq 1$, then $MatchString(org, v_i) = 1$.

The MatchString task also labels the organism with a tag that corresponds to the vector it is best at producing. Specifically, an organism's tag corresponds to the vector whose MatchString task had the highest quality.

The second type of task, CompleteSet, is associated with an unlimited resource that rewards organisms for producing complete sets. Specifically, the task quality is equal to four times the number of complete sets plus the number of additional copies of either string. Task quality is partially defined based on strings that cannot be used as part of a complete set to reward organisms for production quantity. Formally, let c_{low} be the counter with the lowest balance, bal_{low}. Let c_{high} be the counter with the higher balance, bal_{high}. The formula for computing the reward of the CompleteSet function is:

$$CompleteSet(org) = (4 * bal_{low}) + (bal_{high} - bal_{low})$$

4.3 Experimental Setup

Our experiments all used a common set of configurations. Each experiment comprised 20 replicate runs to account for the stochastic nature of the evolutionary process. Each run had 3,600 cells (and thus a maximum population size of 3,600 organisms) arranged in a toroidal topology. Each run started with two untagged asexual self-replicating organisms that constructed a string of the appropriate length for the variant of the problem being attempted. One organism constructed a random string and the other constructed the complement of the random string. Seeding the organisms with strings assists the organisms in solving the bit string production problem because, to receive the resources associated with completing a MatchString task, an organism must construct a string that has over 50% of the bits correct.

The starting length of the organisms was 100 instructions. When an organism replicated, each instruction had a 0.0075 probability of mutating as it was copied. Additionally, each genome had a 0.05 probability of an insertion mutation and a 0.05 probability of a deletion mutation. The offspring of an organism was placed in a random location in the torus. This *mass action* replacement strategy ensures that neighboring organisms are unrelated and thus cooperation would not be simply because of kin selection [1, 5]. The runs lasted for 50,000 updates, where an *update* is the standard unit of time in AVIDA and corresponds to the time required for each organism to execute, on average, 30 instructions.

5. EXPERIMENTAL RESULTS

In this section, we demonstrate our problem decomposition technique on the bit string production problem. First, to ensure that organisms are able to dynamically cooperate to solve the problem, we require all organisms to be specialists and exploit the information produced by other organisms to cooperate. Because the specialist strategy incurs the overhead of cooperation, a generalist strategy may outperform the specialist strategy on simple problems. To test this, we compare the performance of a population of specialists and a population of generalists on the bit string production problem. We then enable organisms to be generalists or specialists and demonstrate that AVIDA varies the population composition in response to the complexity of the problem.

5.1 Evolving Specialists

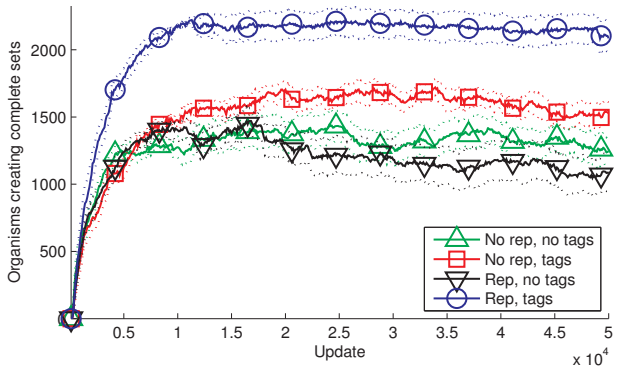
Our first set of experiments test whether AVIDA can evolve a population of specialists that cooperate to solve binary string production problems of varying complexity (i.e., 4-bit, 8-bit, 16-bit, and 24-bit strings). We force all organisms to be *strict specialists*, meaning that they produce strings that match at most one string and therefore must cooperate to create complete sets. To ensure all organisms are strict specialists, we modified the prod-string instruction so that an organism can produce copies only of the string that it is tagged with. Thus, if an organism is tagged with string 0000, then it can only produce copies of string 0000 and must receive string 1111 as a donation.

Although we provide the instructions necessary for the organisms to cooperate using indirect reciprocity, the population must evolve an algorithm that effectively uses these instructions to produce complete sets. It is possible that a population could cooperate using a different technique. To isolate the critical information necessary to enable cooperation among unrelated specialists, we ran four treatments that varied the ability of the organisms to sense tags and reputations. Specifically, the four treatments are:

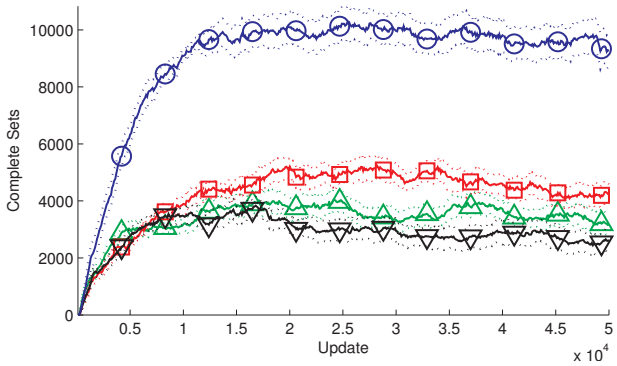
- Organisms do not have the ability to sense either tags or reputation. (no rep, no tags)
- Organisms have the ability to sense tags, but not the ability to sense reputation. (no rep, tags)
- Organisms have the ability to sense reputation, but not the ability to sense tags. (rep, no tags)
- Organisms have the ability to sense both reputation and tags. (rep, tags)

We ran 20 replicates for each variant of the binary string production problem and evaluated the results according to the number of complete sets produced and the number of organisms that produced complete sets. The relative ranking of the treatments was the same for all variants. Figure 3 depicts the results of varying the ability of the organisms to sense tags and reputations for the 16-bit variant. Each treatment produced significantly different results ($p < 0.05$, Mann-Whitney U test). Specifically, Figure 3 (a) depicts the number of organisms that produced complete sets and Figure 3 (b) depicts the number of complete sets produced by the organisms. The solid lines are the mean of 20 treatments and the dotted lines are one standard error of the mean. Overall, organisms without either tag or reputation information (green line with triangles) performed poorly; whereas, organisms with both tag and reputation information (blue line with circles) achieved both the largest number of organisms producing complete sets($\sim$2,000) and also the largest number of complete sets produced ($\sim$10,000).

Counterintuitively, organisms with reputation information, but not tag information (black line with upside down triangles) produced fewer complete sets than organisms without any information. We performed further investigations and found out that, although these organisms are producing fewer complete sets, their average fitness exceeds that of organisms without any information. Essentially, their fitness is greater because they are producing more copies of a single string. Additionally, organisms with tag information, but not reputation information (red line with squares) performed slightly better than organisms with no information, but not nearly as well as organisms with both reputation and tag information ($p < 0.05$, Mann-Whitney U test).



(a) Number of organisms that produce complete sets



(b) Number of complete sets produced

Figure 3: Number of organisms that produce complete sets (top) and the number of complete sets produced (bottom) with four different treatments that vary the tagging and reputation information available for the 16-bit variant. Populations where organisms have both sources of information significantly outperform populations that lack either.

Based on these experiments, we conclude that the specialist strategy is affected by the organism's ability to sense both tags and reputation. For the remainder of the experiments that we describe in this paper, organisms can sense both reputation and tag information.

5.2 Comparing Isolated Populations of Generalists and Specialists

The primary motivation for using a problem decomposition approach is to address problems of increased complexity that are difficult for a standard EA to address. Next, we compare the performance of strict specialists to *strict generalists*, where strict generalists do not have a donate-string instruction and thus are unable to cooperate. To create a complete set, a strict generalist must first create one string (e.g., 0000) in its buffer, produce copies of the string, create the other string in its buffer (e.g., 1111), and then produce copies of it.

Our dual hypotheses for these experiments are that:

1. If a problem is simple, then a population of generalists will outperform a population of specialists, since it is easier to produce both strings than to cooperate. Thus, for simpler variants of the problem, the generalists treatments should both have more organisms that produce complete sets and have more complete sets produced than the specialist experiments.
2. If the problem is complex, then the specialists will outperform the generalists, since it is easier to cooperate than to produce both strings. Thus, the specialist experiments should both have more organisms that produce complete sets and have more complete sets produced than the generalist experiments for the more complex variants of the bit string production problem.

We tested these hypotheses by running 20 replicates of strict generalists and 20 replicates of strict specialists for each of the four variants of the bit string production problem. The results of the experiments are depicted in Figure 4, where the red lines with squares represent the generalists and the blue lines with circles represent the specialists. For the simpler 4-bit and 8-bit variants, the generalists outperform the specialists both in terms of the number of organisms producing complete sets and also in terms of the number of complete sets produced ($p < 0.05$, Mann-Whitney U test). However, for the more complex 16-bit and 24-bit variants, the specialists outperform the generalists ($p < 0.05$, Mann-Whitney U test). The performance of the specialists slightly increases with the complexity of the problem. Specifically, approximately the same number of specialist organisms (∼2000) produced approximately the same number of complete sets (∼10,000) for the 8-bit, 16-bit, and 24-bit treatments. In contrast, the success of the generalists was dependent upon the complexity of the problem. For the 4-bit and 8-bit treatments, ∼2,400 generalist organisms produced ∼20,000 complete sets, but for the 16-bit and 24-bit treatments, ∼400 generalists produced ∼4,000 complete sets. These results support our hypotheses that complex problems favor specialists, whereas simpler problems favor generalists.

5.3 Mixed Populations of Generalists and Specialists

Ideally, a problem decomposition approach would evolve generalists or specialists depending on the complexity of the problem. To test the ability of our technique to automatically vary population composition, we enable both types of organisms to coexist within a population.

Our hypothesis for the next set of experiments is: Given a population in which organisms could be either generalists or specialists, the composition of the population will change depending on the complexity of the problem. Specifically, if the problem is simpler, then the generalists should dominate; if the problem is more complex, then the specialists should dominate. However, enabling organisms to be either generalists or specialists should not negatively affect the ability of the population to solve the problem.

To test this hypothesis, we ran 20 replicates for each of the bit string production problem variants in which organisms could be either generalists or specialists. Specifically, we both enabled the organisms to donate and did not limit them to producing one string. For these results, we consider an organism to be a specialist if it produces only one string and to be a generalist if it produces both.

The green line with triangles in Figure 4 depicts the results of a mixed population of generalists or specialists. In general, the performance of the mixed population toward the end of the run is equivalent to that of the isolated treatment (either generalist or specialist) that performs the best.

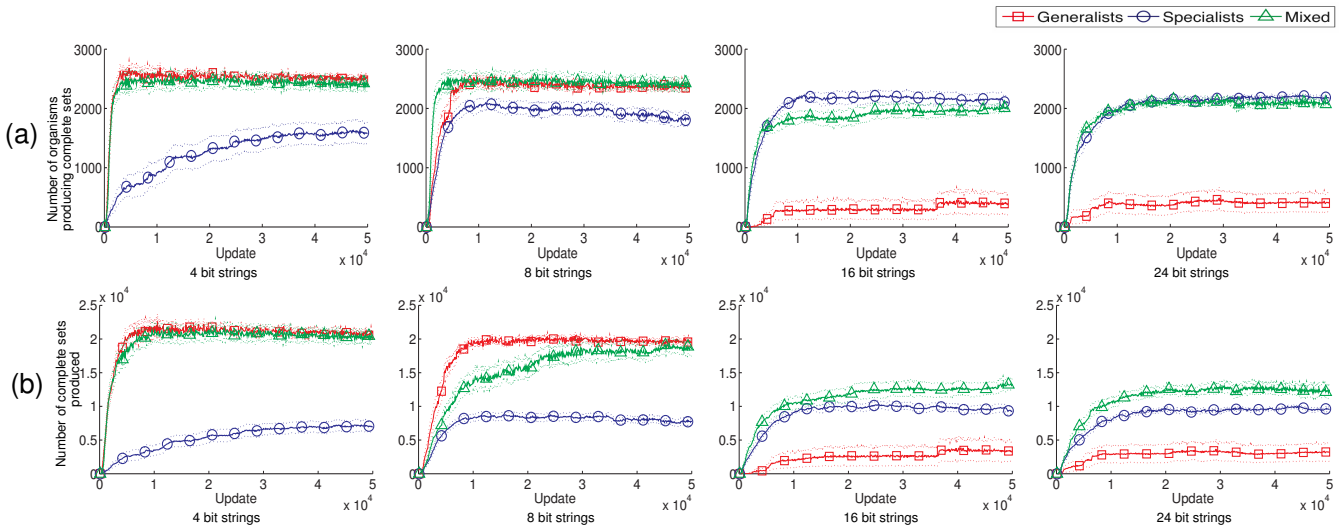


Figure 4: The number of organisms producing complete sets and the number of complete sets produced by generalists, specialists, and mixed populations of both generalists and specialists across the four bit string production problem variants. For simpler problems, generalists outperform specialists; whereas, for more complex problems, specialists outperform generalists. Mixed populations tend to perform at least as well as the isolated populations of generalists or specialists.

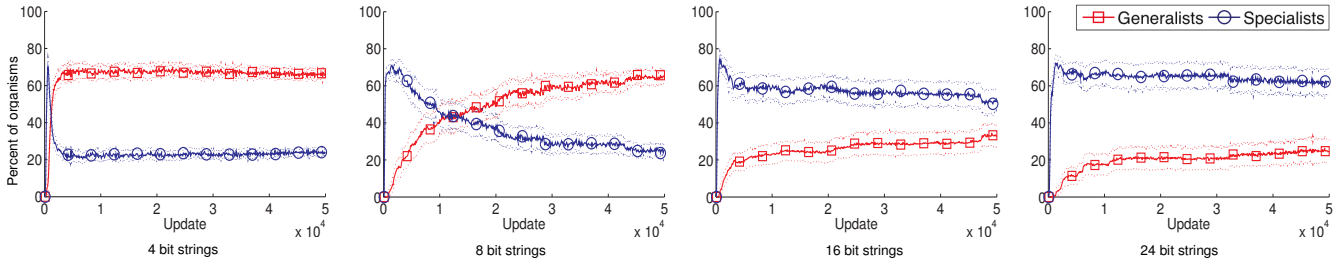


Figure 5: The population composition of mixed populations comprising both generalists and specialists for the four variants of the bit string production problem. In general, the more simple variants have a predominantly generalist population and the more complex variants have a predominantly specialist population.

Additionally, the number of complete sets produced by the mixed populations for the 16-bit and 24-bit variants is higher than that of the isolated populations of both specialists and generalists ($p < 0.05$, Mann-Whitney U test). Therefore, we conclude that the mixed population does not negatively affect Avida's ability to solve the problem.

The average population compositions from the four variants are depicted in Figure 5, where the red lines with squares represent the percent of organisms that are generalists and the blue lines with circles represent the percent of organisms that are specialists. As the complexity of the problem increases, so does the prevalence of specialists. Specifically, in the 4-bit and 8-bit variants, generalists tend to dominate the population ($p < 0.05$, Mann-Whitney U test). However, in the 16-bit and 24-bit variants, the specialists tend to dominate the population ($p < 0.05$, Mann-Whitney U test).

Although the average population compositions make it appear as if the number of generalists in a population gradually increases, for the individual replicates, once a generalist strategy for producing complete sets is found, it quickly sweeps the population. Thus, in many cases, a specialist strategy tended to thrive early on, probably because it is simpler to evolve, but was quickly replaced when a generalist strategy evolved. Three possible factors that could influence the ability of generalist strategy to sweep the population are: First, because generalists do not rely on others for donations of subcomponents, they are a more robust strategy. Second, generalist organisms are, most likely, stingy in the sense that they do not donate. Figure 6 depicts the percent of *altruists* (organisms that donate a string) in the population, the percent of specialists, and the percent of generalists for the 16-bit variant with a mixed population. The percent of altruists almost perfectly tracks the percent of specialists, but appears unrelated to the percent of generalists. Because generalists do not donate, as the number of generalists increases, it becomes increasingly challenging for the specialists to find neighboring organisms with which to cooperate. Third, generalists likely benefit from some donations from specialists, since a generalist is indistinguishable from a specialist that has not yet donated in that both have a reputation of 0.

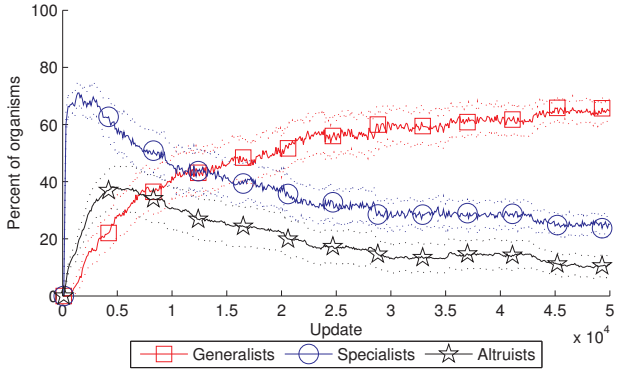


Figure 6: The percent of organisms that are generalists, specialists, and altruists. The percent of altruists in the population closely tracks the number of specialists in the population, indicating that generalists are not making donations.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a digital-evolution based approach to problem decomposition. Our approach enables individuals to be either generalists that independently solve the problem or specialists that each contribute to solving the overall problem. We have demonstrated the feasibility of this technique using several variants of the binary string production problem. As the complexity of the problem increases, the population composition automatically changes from predominantly generalists to predominately specialists. However, once a generalist strategy evolves within the population, it quickly dominates.

Our next step in this work is to increase the complexity of the problem by adding additional strings or longer strings. Additionally, we will apply our approach to problems that require complicated steps to assemble the overall solution, such as with complex arithmetic formulae. To apply this approach to arithmetic problems, we will expand the AVIDA infrastructure to enable organisms to specialize in performing a mathematical operation (e.g., addition) and to donate the results of the operation (e.g., the sum). An organism's reputation will be used to indicate both the quality and quantity of the results it donates. An organism's tag will reflect the operation it has specialized in. Finally, we are investigating the effect of changing from the explicitly designed reputation framework we provided to an evolved reputation and tag system.

7. REFERENCES

[1] J. Clune, H. J. Goldsby, C. Ofria, and R. T. Pennock. Digital evolution confirms and informs the inclusive fitness theory prediction that selection favors increasingly accurate altruism targeting mechanisms. In preparation.

[2] T. Cooper and C. Ofria. Evolution of stable ecosystems in populations of digital organisms. In *Eighth International Conference on Artificial Life*, pages 227–232, Sydney, Australia, 2002.

[3] D. Cornforth and M. Kirley. Cooperative problem solving using an agent-based market. In *Genetic and evolutionary computation conference (GECCO)*, pages 60–71, Seattle, Washington, 2004.

[4] M. Dorigo and M. Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press/Bradford Books, 1998.

[5] S. Goings, J. Clune, C. Ofria, and R. T. Pennock. Kin-selection: The rise and fall of kin-cheaters. In *Ninth International Conference on Artificial Life*, pages 303–308, Boston, MA, 2004.

[6] S. Goings and C. Ofria. Ecological approaches to diversity maintenance in evolutionary algorithms. In *IEEE Symposium on Artificial Life (To Appear)*, Nashville, TN, 2009.

[7] F. J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *In Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1356–1361, 1999.

[8] R. E. Lenski, C. Ofria, T. C. Collier, and C. Adami. Genome complexity, robustness, and genetic interactions in digital organisms. In *Nature*, volume 400, pages 661–664, 1999.

[9] R. E. Lenski, C. Ofria, R. T. Pennock, and C. Adami. The evolutionary origin of complex features. In *Nature*, volume 423, pages 139–144, 2003.

[10] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Genetic and evolutionary computation conference (GECCO)*, pages 363–370, Atlanta, Georgia, 2008.

[11] M. A. Nowak. Five rules for the evolution of cooperation. *Science*, 314(5805):1560–1563, December 2006.

[12] C. Ofria and C. Adami. Evolution of genetic organization in digital organisms. In *Proc. of DIMACS workshop Evolution as Computation*, pages 167–175, Princeton, NJ, 1999.

[13] C. Ofria, C. Adami, and T. C. Collier. Design of evolvable computer languages. *IEEE Transactions in Evolutionary Computation*, 6:420–424, 2002.

[14] C. Ofria and C. O. Wilke. Avida: A software platform for research in computational evolutionary biology. *Journal of Artificial Life*, 10:191–229, 2004.

[15] M. A. Potter and K. A. DeJong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.*, 8(1):1–29, 2000.

[16] R. Sugden. *The economics of rights, co-operation and welfare*. Oxford: Basil Blackwell, 1986.

[17] R. Sun and D. Qi. Marlbs: Team cooperation through bidding. In *International Journal of Computational Intelligence Research*, 2005.

[18] C. O. Wilke, J. L. Wang, C. Ofria, C. Adami, and R. E. Lenski. Evolution of digital organisms at high mutation rate leads to survival of the flattest. In *Nature*, volume 412, pages 331–333, 2001.

[19] C. H. Yong and R. Miikkulainen. Cooperative coevolution of multi-agent systems. Technical report, University of Texas at Austin, 2001.