The Gopher Grounds: Testing the Link between Structure and Function in Simple Machines

Anshul Kamath[1,†][a], Jiayi Zhao[2,†][b], Nick Grisanti[1][c] and George D. Montañez[1][d]
[1]*AMISTAD Lab, Department of Computer Science, Harvey Mudd College, Claremont, CA, U.S.A.*
[2]*Department of Computer Science, Pomona College, Claremont, CA, U.S.A.*

Keywords: Structure and Function, Agents, Genetic Algorithms.

Abstract: Does structure dictate function and can function be reliably inferred from structure? Previous work has shown that an artificial agent's ability to detect function (e.g., lethality) from structure (e.g., the coherence of traps) can confer measurable survival advantages. We explore the link between structure and function in simple combinatorial machines, using genetic algorithms to generate traps with structure (coherence) and no function (no lethality), generate traps with function and no structure, and generate traps with both structure and function. We explore the characteristics of the algorithmically generated traps, examine the genetic algorithms' ability to produce structure, function, and their combination, and investigate what resources are needed for the genetic algorithms to reliably succeed at these tasks. We find that producing lethality (function) is easier than producing coherence (structure) and that optimizing for one does not reliably produce the other.

1 INTRODUCTION

Recent research has shown that *intention perception* (the ability to detect the intentions of others) can benefit artificial agents in a variety of adversarial situations (Maina-Kilaas et al., 2021a; Maina-Kilaas et al., 2021b; Hom et al., 2021). One study found that when artificial agents were able to perceive their environments as intentionally designed (e.g., to detect traps), their survival rates significantly increased (Hom et al., 2021). Within their framework of virtual gophers and projectile traps, Hom et al. assumed that traps designed to harm gophers were much more likely to be "coherent" than those generated uniformly at random, and used this coherence as a basis for creating statistical hypothesis tests (classifying traps as `designed to harm` or `unintended`). Implicit within their experimental framework was an assumed correlation between the structure of a trap and its lethality: coherence implied design which implied lethality. In this study, we test whether the correlation between coherence (structure) and lethality (function) was isolated to their particular context, or whether the relationship holds more generally.

The relationship between structure and function is of general importance to science and engineering, as every machine has structure that at least partially determines its function. In Hom et al.'s work, coherence (a property of trap structure) served as a reliable indicator of trap functionality, able to separate human-designed traps from those generated uniformly at random. While Hom et al. considered only two types of trap-generating processes, we explore the relationship between trap coherence and trap functionality more broadly, for machines generated by processes such as genetic algorithms.

Genetic algorithms are an easy-to-implement metaheuristic method commonly used to solve optimization and search problems. They employ biologically inspired operators such as mutation, crossover and selection (Golberg, 1989; Mitchell, 1998; Reeves and Rowe, 2002). Genetic algorithms can solve problems defined over complex, high-dimensional, multimodal, and/or discrete spaces. They have become a popular optimization tool for *structural optimization*, which is the automated synthesis of mechanical components based on structural considerations (Chapman, 1994; Wang et al., 2006). Generating traps with desirable traits, namely coherence and lethality, can be considered as a simplified structural optimization

[a] https://orcid.org/0000-0002-1641-4430
[b] https://orcid.org/0000-0003-2497-120X
[c] https://orcid.org/0000-0002-6987-9194
[d] https://orcid.org/0000-0002-1333-4611
[†]Denotes equal first-authorship.

Kamath, A., Zhao, J., Grisanti, N. and Montañez, G.
The Gopher Grounds: Testing the Link between Structure and Function in Simple Machines.
DOI: 10.5220/0010900900003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 2*, pages 528-540
ISBN: 978-989-758-547-0; ISSN: 2184-433X

problem over a discrete search space, motivating their use in the present study.

We examine the correlation between trap coherence and trap lethality for structures produced by genetic algorithms and find that Hom et al.'s implicit assumption may not hold in all situations. Furthermore, we investigate genetic algorithms' capacity to generate functional and coherent traps, given that humans have little difficulty producing them. Finally, to the extent our genetic algorithms are successful, we determine what information resources are necessary for them to generate traps with either lethality or coherence (or both), by varying the information encoded in their fitness functions. We find that contrary to assertions by earlier researchers (e.g., (Häggström, 2007)), simply having fitness functions with structural order (such as local neighborhood clustering) is not sufficient for genetic algorithms to perform the search task any better than blind uniform sampling. To the contrary, we find that *alignment* of the fitness functions to the particular task at hand is also necessary, agreeing with other prior work (Montañez, 2017; Montañez et al., 2019).

2 RELATED WORK

There has been a wide range of research on the relationship between structure and function. Gero and Kannengiesser proposed the Function-Behaviour-Structure ontology (Gero and Kannengiesser, 2007), which provides three categories for the properties of a human-designed object: Function (F), Behavior (B), Structure (S). They assert that function (F) is "ascribed to" behaviour (B) by establishing a teleological connection between a human's design purpose and what the object does, while behaviour (B) is "derived from" structure (S) directly. Thus, they imply there is no direct connection between function and structure.

Bock and Wahlert, however, argue that structure and function constitute the two inseparable dimensions of biological features when considering morphology and evolutionary biology, and "must always be considered together" (Bock and Wahlert, 1965). Weibel proposed the *theory of symmorphosis*, which predicts that the design of parts within an organism must be matched to their functional demands. He also tested these predictions by quantifying the relationship between structure and function in different organ systems (Weibel, 2000).

While Hom et al. only dealt with two kinds of traps (those generated uniformly at random and those they designed to be lethal (Hom et al., 2021)), we explore the lethality and coherence produced by a wider variety of trap-generating processes. In particular, we generate traps with structure (coherence) and no function (lacking lethality), traps with function and no structure, and traps with both structure and function, by means of genetic algorithms.

Genetic algorithms compose part of the larger field of evolutionary optimization (Golberg, 1989; Mitchell, 1998; Reeves and Rowe, 2002). Evolutionary optimization has been applied to a diverse range of problems, with their associated difficulties. One such difficulty is uncertainty—fitness functions are often noisy or approximated, environmental conditions change dynamically, and optimal solutions may change over time (Krink et al., 2004; Then and Chong, 1994; Bhattacharya et al., 2014). A wide range of techniques have been developed to combat such problems (Jin and Branke, 2005).

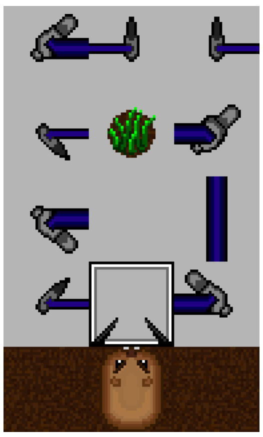
3 METHODS

3.1 Traps

We adapt the virtual trap framework of Hom et al. (Hom et al., 2021). A trap contains food to entice an artificial gopher to enter and laser beams to "kill" the gopher once it is inside. The traps are simple, combinatorial objects embodying both form and function. This allows us to explore that relationship in a controlled manner while investigating the effect of genetic algorithm processes on their generation.

Each trap consists of a 4×3 grid. Three of the tiles are fixed for all traps: one acts as an entrance and senses when the gopher enters the trap; directly above the entrance is a fixed blank floor tile that the gopher can traverse to get to the third fixed tile, which holds the food enticing the gopher to enter the trap. The remaining nine tiles can either be blank floor, laser emitters (arrows), or wires meant to connect the laser emitters to the sensor at the entrance. The arrows and wires can be one of three thicknesses, with elements of greater thickness more likely to kill a gopher. In particular, the probabilities of killing a gopher on a successful hit with a wide, normal, skinny arrow are $P_{k,w} = 0.45$, $P_{k,n} = 0.3$, and $P_{k,s} = 0.15$ respectively. Two example traps are shown in Figure 1.

The laser will only be functional if the thickness of the arrow matches the thickness of every wire piece connecting it to the sensor. A wire tile can either be straight or bent at a right angle, and can be rotated by 90° as needed. Arrows can similarly be rotated by 90°. Accounting for all possible rotations and thicknesses, there are a total of 91 possibilities each of these nine tiles can take. Hence—keeping in mind


(a) Example of a functional trap generated by the genetic algorithm.

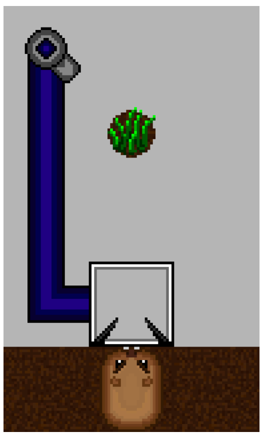

(b) Example of a designed trap created by Hom et al.

Figure 1: Example traps.

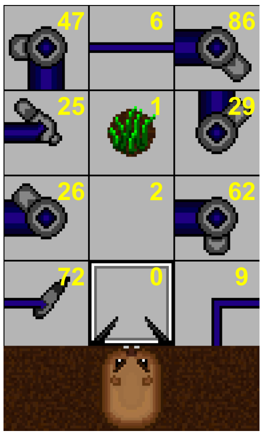
that the entrance and food tiles are each used exactly once per trap—there is a total of $91^9 \approx 4.28 \times 10^{17}$ possible traps in this framework. We let $\mathcal{X}$ denote the set of all valid traps, with $|\mathcal{X}| \approx 4.28 \times 10^{17}$.

In agreement with Hom et al., we define *trap coherence* as follows (Hom et al., 2021). First, we must define the notion of a *coherent connection*. There is a coherent connection between two non-entrance tiles if all of the following conditions are met: (1) both tiles contain either a wire or an arrow, (2) the thicknesses of the two elements match, and (3) the two elements share an endpoint (i.e., the rotation of the wires align). The *coherence* of a trap is then the number of coherent connections per nonempty (wire or arrow) tiles.

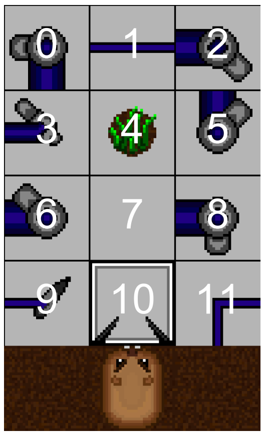
Furthermore, we define the *lethality* of a trap as the probability that it kills a gopher entering it. Also referred to as functionality in this context, lethality can either be estimated by running simulations that present traps to gophers of varying hunger levels and measure the proportion of gophers killed, or computed analytically. We compute both coherence and functionality analytically, with empirical simulation verifying the correctness of our approach (not shown).

3.1.1 Trap Encoding

We introduce a genotypic representation of a trap (called an *encoding*) for our genetic algorithm. Our encoding method takes in a 4×3 trap matrix and outputs a 1×12 array of integers. First, we look at the 93 individual tiles and map each of them to a unique integer $x \in [0, 92]$. For instance, the skinny arrow with right-acute angle rotated at $0°$ has the encoding 33. Note that the encoding for door tile is 0, food tile 1, and the floor tile 2. The encoding for each tile of an example trap is shown in Figure 2a.



(a) Encodings for 12 trap tiles.



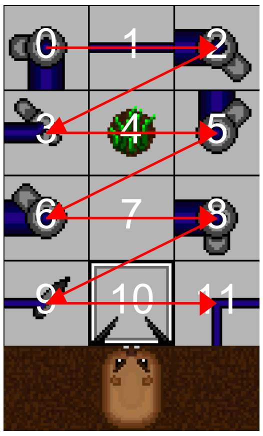
(b) Indexing the 12 trap tiles.

Figure 2: Tile encoding and trap location indexing.

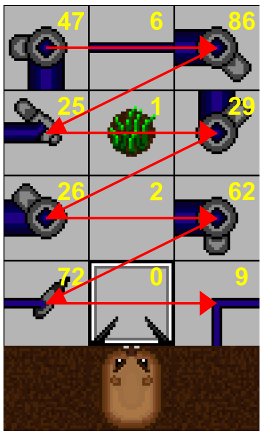
Next, we enumerate the 12 tiles in a trap as shown in Figure 2b. Intuitively, each trap can be encoded by listing the encodings of its tiles in the order $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$, creating a zig-zag pattern; this is Method 1, demonstrated in Figure 3. A potential problem with this method is that the order of the tiles in an encoding does not reflect the spatial layout of the trap. See, for example, how tiles 2 and 3 are positioned on opposite sides of the trap but are ordered sequentially in the encoding. To address this, we developed an additional method, Method 2, which encodes a trap by listing the encodings of its tiles in the order $(9, 6, 3, 0, 1, 2, 5, 8, 11, 10, 7, 4)$. This order lists the traps in a wrap-around pattern, as shown in Figure 4. Thus, sequential tile encodings in the trap encoding are adjacent to each other in the actual trap.

3.2 Gophers

Like Hom et al., we used gophers as the artificial agents in our experiments. Hom et al. designed two types of gophers: *intention gophers* and *baseline gophers*. Intention gophers were given intention perception—the ability to assess the coherence of the trap and then determine if the trap is deliberately harmful, based on its coherence. If the trap is found sufficiently coherent, the intention gopher will avoid it; otherwise, the gopher will enter it. Baseline gophers, however, will enter any trap according to some predetermined probability. Whereas Hom et al. compared survival outcomes of intention and baseline gophers, we seek to merely assess the coherence and lethality of traps generated using a genetic algorithm. As such, we simplify our gopher framework to

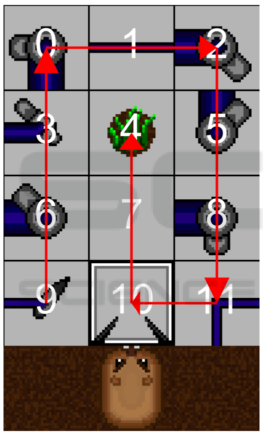


(a) Encoding with permutation (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)

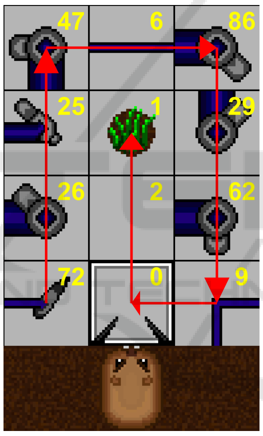


(b) Encoded trap: [47, 6, 86, 25, 1, 29, 26, 2, 62, 72, 0, 9]

Figure 3: First encoding method and corresponding encoded trap, Method 1.



(a) Encoding with permutation (9, 6, 3, 0, 1, 2, 5, 8, 11, 10, 7, 4)



(b) Encoded trap: [72, 26, 25, 47, 6, 86, 29, 62, 9, 0, 2, 1]

Figure 4: Second encoding method and corresponding encoded trap, Method 2.

include only one type of gopher, which is described below. Each gopher has a hunger level H uniformly sampled from $\{0, 0.2, 0.4, 0.6, 0.8\}$, and will enter a trap with probability

$$P_e'(H) = P_e \cdot (1 - H^{10}) + H^{10},$$

where P_e is the default probability of entering (what a baseline gopher would have in Hom et al.'s framework) and $H \in [0, 1)$ is the current hunger level.

The eating time t_{eat} of the gopher is how long (in discrete time steps called *frames*) the gopher takes to eat and is selected according to the probability vector

$$\vec{p} = [p_1, p_2, p_3, p_4, p_5],$$

where, p_j is the probability that a gopher eats for j frames. By this definition of $\vec{p}$, each gopher will take between 1 and 5 frames (inclusive) to eat. The probability vector $\vec{p}$ depends only on the default probability of entering P_e, and it is calculated by methods in (Hom et al., 2021).

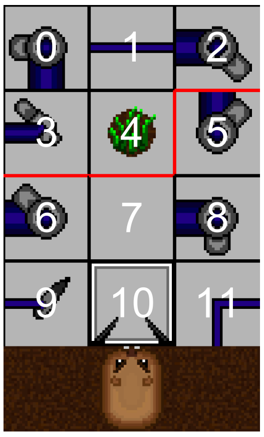
When a gopher decides to enter a trap, it will directly head to the food with speed 1 tile/frame, eat for t_{eat} frames, and then exit the way it came with speed 1 tile/frame. When the gopher first enters the trap, the sensor will detect it instantly and release a pulse on each side of the entrance door. If the arrow is coherently connected to the door, the pulse will travel through the wire with speed 1 tile/frame and fire a projectile with speed 1 tile/frame that can possibly hit the gopher. We designed our gophers to be "skittish," that is, they flee traps whenever a laser fires, regardless of their hunger level.

3.3 Genetic Algorithms

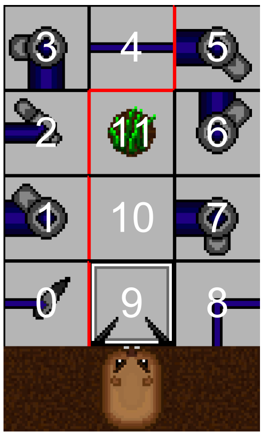
In mimicking natural selection, genetic algorithms optimize toward increasing fitness. Here the population consists not of biological organisms but of potential solutions to an optimization problem.

A genetic algorithm requires a fitness function (see below) to generate a subset of elements $X \subseteq \mathcal{X}$. In our setting, for example, $\mathcal{X}$ is the set of all valid traps defined in Section 3.1. Then, the algorithm goes through a process of selection, recombination, and mutation to generate a new subset $X' \subseteq \mathcal{X}$. The algorithm usually begins with a generation of randomly generated elements. For each generation of the genetic algorithm, a subset of the population is selected to "reproduce" to fill the next generation's population. Much as organisms of higher fitness in nature are more likely to reproduce (by definition), elements of higher fitness in the genetic algorithm are more likely to contribute to the next generation. To achieve this, we used a roulette-wheel selection process: two elements of the previous generation are selected at random, with their probability of selection directly proportional to their fitnesses. These two elements undergo a recombination process in which their information is combined—usually by splicing each element in two and sampling a piece from each element—to produce a new element that contains information from both parents. Example of such splits are shown in Figure 5.

Finally, we mutate the resulting element by flipping a random element in its encoding to some random value in the cell alphabet, in the style of a genetic point mutation. This element is now fully formed and joins the new generation. The process repeats until the size of the new generation matches that of the old gen-



(a) Method 1



(b) Method 2

Figure 5: An example of recombining at the fifth cell under both encoding methods.

eration. New generations are iteratively created until some threshold is met, usually requiring that one of the elements have a fitness above a specified value or that a specific number of generations is produced. The single element with highest fitness is then returned. Algorithm 1 provides pseudocode for this entire process, where $f(\cdot)$ is the fitness evaluation function.

Algorithm 1: Sample Genetic Algorithm.

```
 1: procedure GENETICALGORITHM
 2:     globalBest ← none
 3:     population ← generateRandomPopulation()
 4:     while not terminationConditionsMet do
 5:         newPopulation ← empty
 6:         for element in population do
 7:             selectedPair ← roulette(population)
 8:             combined ← recombine(selectedPair)
 9:             mutated ← pointMutate(combined)
10:             newPopulation.add(mutated)
11:         popBest ← bestTrap(newPopulation)
12:         if $f(popBest) > f(globalBest)$ then
13:             globalBest ← popBest
14:         population ← newPopulation
15:     return globalBest
```

3.4 Fitness Functions

To determine the optimality of a member $x \in \mathcal{X}$, we define a **fitness function**. This function $f : \mathcal{X} \to \mathbb{R}$ allows us to impose an ordering on the elements of the search space, $\mathcal{X}$, thereby giving the algorithm a measure of how optimal the current solution is. We next describe our process for creating a set of fitness functions based on different optimization criteria. To make our language more precise, we define the order and alignment of a fitness function. **Order** within a fitness function refers to any underlying patterns or regularities that may be present in its spatial distribution of values, such as local neighborhood clustering. The **alignment** of a fitness function refers to the measure of preference for elements in a specific target set of interest. For example, if we want to optimize for coherence, a fitness function is correctly aligned if it gives high fitness values to coherent traps and low fitness values to non-coherent traps.

In the subsections that follow, we describe the fitness functions used in our experiments.

3.4.1 Random Fitness

For any trap $x \in \mathcal{X}$, we define the random fitness function as $r(x) = n$ where n is a number chosen uniformly at random from the range $[0, 1)$. In other words, when running the genetic algorithm, we impose a random ordering on the population through the random fitness function and select traps according to this random ordering. Thus, we do not expect the random fitness function to exhibit either order or alignment.

3.4.2 Binary Distance (Hamming) Fitness

Let $t \in \mathcal{X}$ be a uniformly sampled random trap, and let it serve as our target trap. The binary distance fitness function returns a number indicating how “close” a given trap to the target trap t.

For any trap $x \in \mathcal{X}$, we define the binary distance fitness to be

$$d(x) = \frac{\#_{\text{diff}}}{\#_{\text{total}}},$$

where $\#_{\text{diff}}$ is the number of differences in the tiles between x and t, and $\#_{\text{total}}$ is the maximum number possible differences; this value is $12 - 3 = 9$ as, it excludes the three fixed cells—door, floor, and food—in the middle column of every trap. Thus, this fitness function will quantify how close x is to t, thereby giving it local neighborhood structure and order. However, this measure does not give us any meaningful information about the coherence or lethality of the trap. Hence, this fitness function is not aligned.

3.4.3 Functional Fitness (Lethality)

For any trap $x \in \mathcal{X}$, we define the functional fitness by

$$f(x) = \frac{P_{kill}(x)}{P_{max}},$$

which is the normalized probability that the trap x will kill the gopher. $P_{kill}(x)$ is the (non-normalized) probability that the trap x will kill the entering gopher, and P_{max} is the maximum probability of a gopher dying.

We calculate these values analytically (as detailed in Appendix A).

Our fitness function $f(x)$ evaluates the functionality of traps in terms of killing gophers. Since it gives higher values to lethal traps and similar traps often have similar lethality, this fitness function has both order and alignment.

3.4.4 Coherent Fitness

Following Hom et al., for any trap $x \in \mathcal{X}$, we define the coherent fitness function as

$$g(x) = \frac{c_x}{t_x},$$

where c_x is the total number of coherent connections and t_x is the number of nonempty cells (Hom et al., 2021). Since $g(x)$ gives higher values to coherent traps and similar traps have similar coherence, this fitness function has both alignment and order, as well.

3.4.5 Multiobjective Fitness

For clarification, we will define two different fitness functions in this section. Both of them aim to prioritize traps that are both lethal and coherent, penalizing large gaps between coherence and lethality values (so that the process does not merely optimize for lethality at the expense of coherence, for example).

We define a *global multiobjective fitness function* $h : \mathcal{X} \to \mathbb{R}$ which takes a trap as an input and outputs a fitness value representing both coherence and lethality of the trap. This fitness function is similar to functional and coherence fitness functions, but it is not efficient enough when optimizing multiple objectives. Therefore, we implement an additional *local multiobjective fitness function*, $\varphi : \mathcal{X}^n \to \mathbb{R}^n$, where $n \in \mathbb{N}$ is the population size. The function takes in a population of traps and outputs a fitness value for each, effectively imposing a partial ordering on the current population. The local multiobjective fitness function only provides relative ranks of traps within a generation; the same trap can be assigned different fitness values in different generations, making it difficult to compare traps across generations, which was necessary for our analysis.

Therefore, we use the *global* multiobjective fitness value of each trap for record-keeping and analysis, and use the *local* multiobjective fitness function for selection within the genetic algorithm, for efficiency. We describe both fitness functions in detail next.

Local Multiobjective Fitness. We begin by defining a notion of dominance among traps. Trap A *dominates* trap B if trap A has both a greater functional fitness and a greater coherent fitness than trap B.

Our local multiobjective fitness function is defined as follows. Each trap in the population is given a base score, which is equal to one more than the number of traps it dominates. Next, to disincentivize sampling traps that are too similar to each other, we compute the point-wise Euclidean distance between each trap and its closest neighbor of the same base score. If there is only one trap of a particular base score, then the distance is defined to be $\sqrt{2}$ (the maximum possible distance between two traps). The point-wise distance for each trap is divided by $\sqrt{2}$ and added to its base score to determine its boosted score. Each boosted score is then divided by the maximum boosted score across the population, leaving the most fit trap with a fitness value of 1. This set of boosted scores—which is contained in the interval $(0,1]$—is then returned.

This function is a variation of a standard method for multiobjective evolutionary optimization (Fonseca et al., 1993). The number of other traps a particular trap dominates is a good measure of overall fitness, as traps with either a greater functional or coherent fitness are likely to dominate more traps. By boosting each trap's score by an amount proportional to its distance from the nearest neighbor of the same base score, the genetic algorithm is disincentivized from sampling traps that are too similar to each other. This should not only help the genetic algorithm reach a fitness threshold more quickly, but encourage populations to become diverse.

Global Multiobjective Fitness. For any trap $x \in \mathcal{X}$, we define the global multiobjective fitness to be

$$h'(x) = \frac{f(x) + g(x)}{e^{2|f(x)-g(x)|}},$$

where $f(x)$ and $g(x)$ are the functional and coherent fitness values of x, respectively. We sought to minimize the difference between $f(x)$ and $g(x)$ in the final generated trap, so we introduced another penalty to disincentivize solely optimizing coherence or lethality alone (which could lead to large gaps between the functional and coherent fitness values):

$$h(x) = \begin{cases} h'(x) & |f(x) - g(x)| \leq k_{diff}, \\ \frac{1}{10k_{diff}} h'(x) & |f(x) - g(x)| > k_{diff}, \end{cases}$$

where k_{diff} is some constant we defined. Furthermore, since this fitness function assigns higher values to traps that are both lethal and coherent, it also has both order and alignment.

4 EXPERIMENTAL SETUP

The general framework for our experiments consists of generating a trap using our genetic algorithm, then assessing the trap to determine what expected proportion of gophers would die when entering it.

To generate a single satisfactory trap—a trap optimized for either random fitness, hamming fitness, coherence fitness, functional fitness, or multiobjective fitness—we create an initial population of size 20 and run the genetic algorithm with its corresponding fitness function for 10,000 generations. Then, we output the trap with the highest fitness value among all 200,000 traps. We repeated this process 1,000 times for each fitness function, performing 1,000 independent trials and thus generating 1,000 optimized traps for each.

Additionally, in order to compare our experiments to some baseline distribution, we randomly generate the same number of traps uniformly at random. We also use the designed trap's given by Hom et al. as a control for highly coherent and lethal traps (Hom et al., 2021).

5 RESULTS

5.1 Proportion Vectors

First, we show the coherence and lethality distributions of traps generated by genetic algorithms with different fitness functions. Figure 6 depicts the proportion of traps at given lethality/coherence levels, separated by each method used to generate the traps.

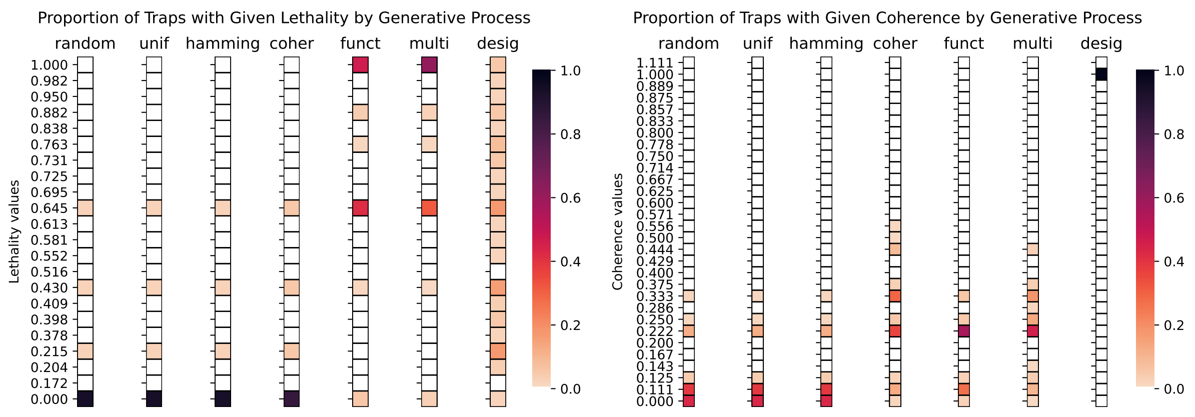
Analyzing these vectors, we notice that the random, uniform, and binary distance methods give identical proportion vectors with respect to both lethality and coherence. This suggests two things; first, that uniform random sampling of traps is as effective as using a genetic algorithm equipped with a fitness function that randomly assigns each trap its fitness value. Second, it suggests that simply having order in a fitness function is not sufficient to improve the probability of success (contrary to (Häggström, 2007)): fitness functions must also have intentional alignment, which leads the genetic algorithm towards a target set of interest. Since the binary distance fitness function chooses a random target trap and tries to generate similar traps, we can see that this fitness function has sufficient order, but it does not align with the problem of optimizing for coherence and/or lethality. In other words, genetic algorithms with random, uniform random, and binary distance fitness functions only perform as well as uniform random sampling in terms of generating coherent and/or lethal traps, due to their lack of correct alignment. More simply, we must embed some selection bias towards the target set into our fitness function to get improved results. It is not sufficient to simply have order.

Next, note that the lethality fitness function is not aligned with the coherence fitness function and vice versa. More specifically, the coherence proportion vector in Fig. 6a is highly similar to the uniform random proportion vector when we optimize for lethality. This is due to a mismatch in alignment. The intersection of the coherence and lethality target sets is too small, so when we optimize for coherent traps, we do not necessarily get lethal traps as well. The same reasoning explains why the lethality proportion vector in Fig. 6b is highly similar to the uniform random proportion vector when we optimize for coherence.

Moreover, notice that a large proportion of traps reached maximum lethality in both the functional and multiobjective experiments in Fig. 6a. While this makes sense (since these fitness functions were meant to optimize for lethality), notice the same cannot be said for coherence in Fig. 6b, whose maximum value (of sufficient magnitude such that it does not appear white on the color bar) is 0.556. Thus, in 10,000 generations we have seen that at least $\sim$40% of traps had maximum lethality when optimizing for lethality, while only $\sim$10% of traps had coherence of 0.556 when optimizing for coherence. This discrepancy suggests that it is much easier to create lethal traps than it is to create coherent traps. It is important to note that the proportion of traps varies greatly. While it appears there are no traps with coherence greater than 0.556, there are, but not in significant enough quantities to show up in Figure 6b.

Additionally, notice that for uniform-random sampling, there were traps sampled with lethality 0.215, 0.430, and 0.645. These all correspond to simple, functional traps such as those shown in Figure 7.

Finally, we can notice that the human designed traps of Hom et al. are starkly different from any traps generated by our programs. Their traps were created to simply zap gophers in a broad variety of ways; they were not intentionally optimized for coherence nor for perfect lethality, but optimized for diverse functionality. However, we see that every designed trap had a coherence of 1, and its lethality was essentially randomly distributed. Thus, we can confirm that the form of a trap is a good indicator of intended construction. It is also important to note that, as far as we are aware, the human-designed traps of Hom et al. were not intentionally designed to be highly coherent. Instead, they were made to test all types of traps, as shown by the uniform probability distribution for lethality in Figure 6a. This unintended coherence could simply be attributed to the fact that humans have an innate sense of structure. However, this affinity for structure



(a) Lethality.

(b) Coherence.

Figure 6: Heat map showing the proportion of traps with a given lethality / coherence, split by the parameter which the genetic algorithm was optimizing for (and designed traps).

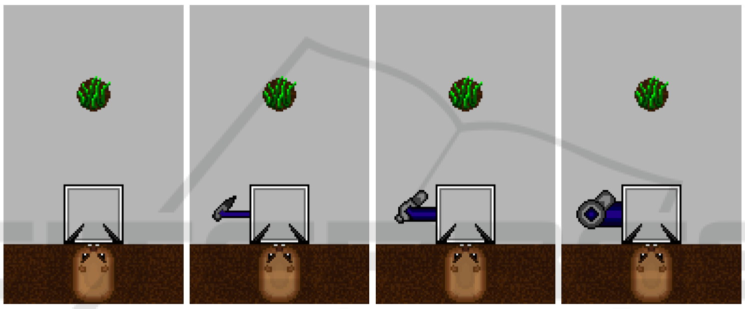


Figure 7: Four simple traps of lethality 0, 0.215, 0.430, and 0.645, respectively.

is challenging to imitate *in silico*, as shown by the lack of coherent traps generated by our algorithms, even when trying to optimize for coherence directly.

5.2 Time to Optimal Trap

Figure 8 shows when an optimal trap was found for each trial. The functional fitness function seems to find its optimal trap in roughly 100 generations. This makes sense, since we have already seen that it is not very hard to get a maximally lethal trap; all that is needed are two angled, thick arrows on either side of the door. Additionally, its interquartile range is extremely small (as shown by the outliers proximity to the box) which provides further evidence of the ease of finding a maximally lethal trap.

The coherence and multiobjective fitness functions are more interesting. They both have similar (essentially the same) distributions, with medians at roughly 3,000 generations (where the coherence distribution is slightly higher), and ranges spanning the whole x-axis. The interquartile range is higher for the multiobjective function.

Figure 9a shows the average optimal fitness across all generations. This is calculated as follows: take a generation *i* in all 10,000 generations. We then take the best observed fitness up to generation *i* for each trial, and then average them across all 1,000 trials. This gives us monotonically increasing averages for all three functions. For the functional fitness function, we notice that the curve approximates a step function. This makes sense, since we know that the functional experiments quickly converge to 1.0. The lack in visibility of the confidence intervals is due to low variance in the outcomes. Notice, however, that the coherence and multiobjective curves converge at a much slower rate. This is due to the increased difficulty in finding coherence. Additionally, we note that these values seem to approach roughly 0.75 and 0.6 respectively, which indicate that these are the highest fitness values we found (on average). Finally, notice how the change in the slope drastically falls off after generation 4,000. This is due to the fact that most of the experiments have converged, and there are few traps with higher fitness left for the algorithm to find.

On the other hand, Figure 9b shows the average population fitness across trials at a given generation,

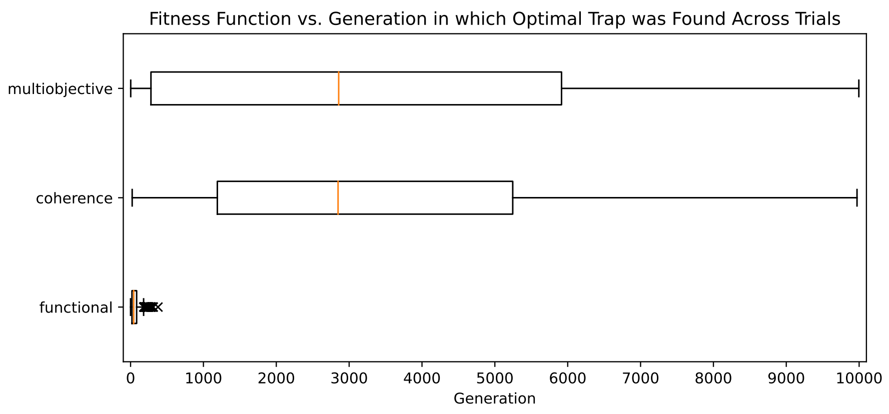
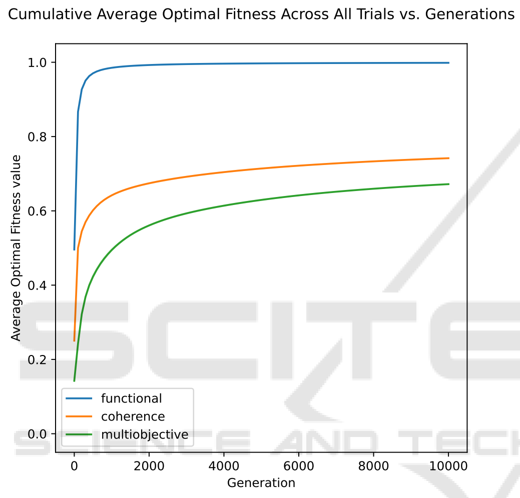
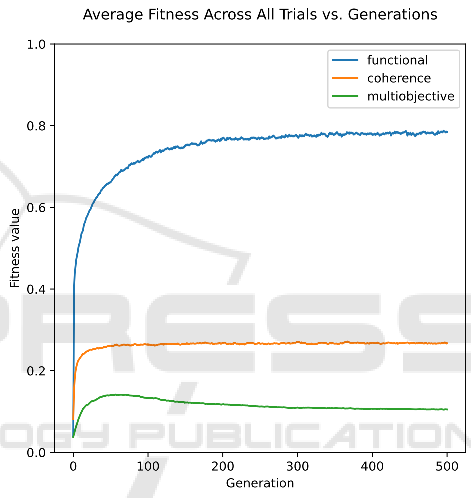


Figure 8: Boxplot showing the distribution of when the optimal trap was found across all generations.



(a) Line plot showing the cumulative average optimal fitness across all trials over generations. The shaded region represents the 95% confidence interval.



(b) Line plot showing the average fitness across all trials over generations. The (imperceptible) shaded region again represents the 95% confidence interval.

Figure 9: Line plots showing the trends of average fitness over generations

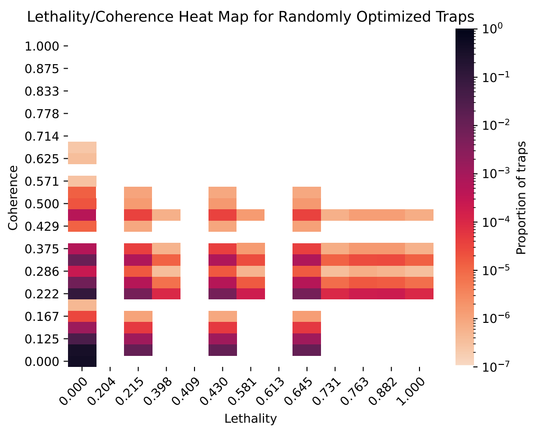
for all generations. Unlike the previous graph, this was made by taking a given generation of the algorithm and averaging all the fitnesses across the 1,000 trials (instead of the *optimal cumulative* fitness). We only plot generations 0-500 since there are no notable deviations from the observed trend. As we can see, the average fitness across all trials seems to plateau for each fitness function. While this makes sense for the functional fitness function (since it arrives at its max value the fastest), we notice that the average fitness for coherence and multiobjective plateau at comparatively low fitness values of 0.25 and 0.1 respectively. This is surprising, since our genetic algorithm is equipped to generate coherent structures; we expected the average fitness across generations to generally increase in the absence of finding the optimal solution.

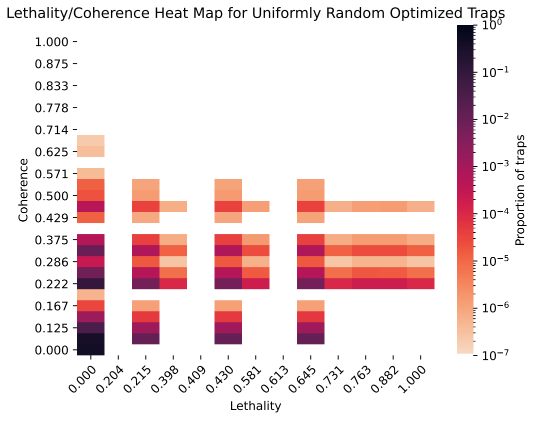
Finally, notice that there is a peak in the multiobjective curve. We believe this is due to the algorithm ceasing to optimize for primarily lethal traps and instead optimize for coherently lethal traps. In this event, the algorithm would have to sacrifice lethality in favor of coherence and lethality, which decreases the overall fitness.

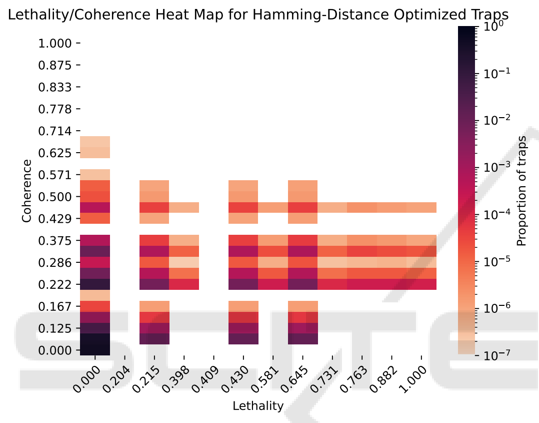
5.3 Frequency Density Heatmaps

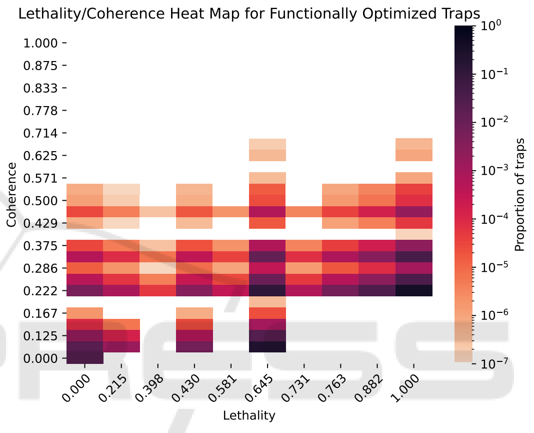
Figures 10a - 10c show the log proportion of traps with a given lethality and coherence value. Note that all traps generated across the 10,000 generations of the genetic algorithm are included in this dataset.

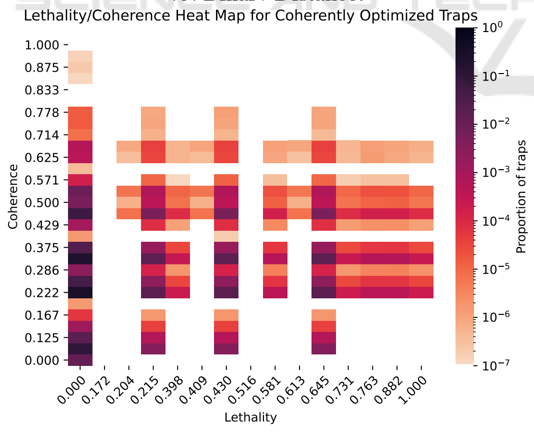
We can observe some shared trends in these heatmaps; first, notice that the most common lethality values are 0, 0.215, 0.430, and 0.645. These all corre-


(a) Random.


(b) Uniform-random.


(c) Binary Distance.


(d) Functional.


(e) Coherence.

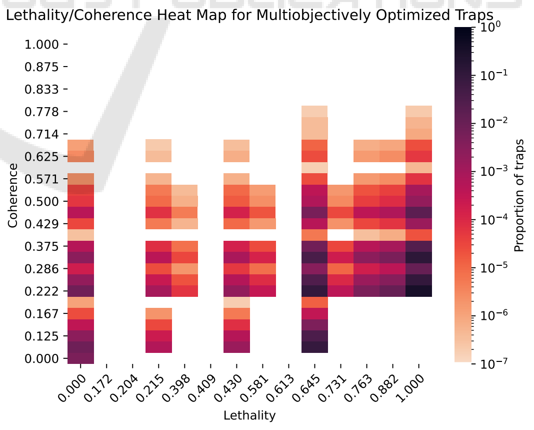

(f) Multiobjective.

Figure 10: Heatmaps representing the log proportion of traps with a given lethality and coherence value separated by optimization parameter.

spond to simple, functional traps such as the examples given in Figure 7.

Since these traps are quite simple, we observe that it is not too difficult for the algorithm to generate lethal traps. To see this more clearly, consider the coherence values below 0.222 (which corresponds to having at most one coherent connection) among all graphs. As is evident, there are only four possible lethality values among these coherences, since any coherence value below 0.222 indicates there is at most one functional arrow. Thus, we can see that lethal traps also provide a baseline coherence, since the only way a trap can be lethal is if it is built upon coherent connections. An absolute lack of coherence implies a lack of functionality.

Studying Figures 10a - 10c more closely, we can see that these heatmaps look roughly the same. Thus, this supports the idea that the random fitness function, uniform-random sampling, and the binary distance function all generate results of similar quality (as evidenced by the way they sample the same space during the genetic algorithm process). More specifically, the random and binary distance functions seem to be as good as uniform random sampling, and they cannot reliably produce coherent or lethal traps.

Finally, studying Figures 10d - 10f, we see that the multiobjective heatmap shares characteristics with both the functional and coherent heatmaps. More specifically, we see that the multiobjective heat map resembles the functional heatmap closely, except that it shifts some frequency mass towards higher coherence traps. This is evidence that the multiobjective heatmap was successfully creating coherent and functional traps, however it does not seem to be able to reliably create highly functional and coherent traps.

6 DISCUSSION

We can see how the complexity of generating coherent traps factored into the multiobjective function. While the multiobjective frequencies in Figure 10f resemble aspects of both the lethality and coherence frequency plots, it is much further skewed towards the lethality frequency plot. This is due to the fact that our genetic algorithm could much more easily optimize for lethality than coherence. We generally see highly lethal traps within 200 generations. Observing these lethality-optimized traps, it is apparent that the genetic algorithm almost always produces traps with the simplest form of lethality: arrows attached to the side of the door. For these traps, a baseline level of coherence is guaranteed as they have two coherent connections to the door. However, it is difficult to generate traps with *just* these two arrows. Instead, most of these functional traps have additional, non-functional tiles that decrease their coherence.

While lethality-optimized traps usually reach maximum lethality within 200 generations, the coherence-optimized traps do not reach their maximum until $\sim$2,500 generations. We can attribute this difference to the lack of a short-cut to coherence; while we can generate extremely simple but functional traps by attaching arrows to the door, coherent traps by nature require coordination.

This may also suggest that there are more functional traps than coherent traps. We can verify this through simple calculations and sampling experiments. Among all possible traps, there are at least $91^7 \approx 2.288 \times 10^{13}$ traps with lethality 1, even if we only consider traps with two wide arrows directly attached to the door, pointing at the middle column. However, there are only 26,733 traps with coherence 1. Moreover, according our sampling of one million uniform random traps, 2.27% of the traps have lethality values greater than or equal to 0.5, but only 0.0037% of the traps have coherence values greater to equal to 0.5.

Also note that the coherence of a trap is solely determined by the connectedness of the wires, which is a property that is not preserved under mutation and recombination. On the other hand, many traps can have arrows in the first cell adjacent to the door, so mutations and recombinations do not affect these traps as much. Hence, as single, stepwise changes may reduce, rather than improve fitness, improvements in trap coherence may require multiple, coordinated changes in trap cells. Therefore, modifying the genetic algorithm to allow coordinated mutations and recombination may be a possible method to overcome the current barriers to achieving high coherence levels.

7 CONCLUSIONS

How strong is the link between coherence and lethality, representing structure and function? Since lethal traps require at least one coherent connection, where there is a lack of coherence, there is a corresponding lack of lethality. However, it's important to note that even though high lethality guarantees a baseline coherence (≥ 0.222) when producing lethality-optimized traps, the genetic algorithm does not generate *high* coherence as a side-product. Similarly, coherence alone does not beget lethality, but it may boost the probability of finding firing traps. So the relationship, while present, is weak.

As we can see in Figure 6b, the designed traps all have a coherence of 1, whereas the genetic algorithm was not able to reliably achieve coherence levels above 0.556 when optimizing for coherence. These findings suggest that contingent coherence remains a plausible sign of intentional construction, as optimizing trap coherence is effortless for human designers. This supports Hom et al.'s use of trap coherence as an indicator of intentional design in their original experiment.

Our genetic algorithms were able to generate certain types of traps well. Specifically, they were efficient in generating highly functional traps, producing maximally lethal traps in a reasonably small number of generations. Coherent traps, however, were much more difficult to produce. The genetic algorithms struggled to produce traps with high levels of coherence, even when optimizing for coherence directly. Optimizing for both lethality and coherence proved even more difficult. Most of the traps generated had high lethality but relatively low coherence.

To reliably generate traps with high coherence and/or lethality using genetic algorithms, both order and correct alignment of fitness functions were essential. First, our fitness functions required order, with neighborhood constraints on the elements of the search space, allowing a genetic algorithm to perform a meaningful local search. However, order in the fitness functions was not enough, as was evident in the failure of the binary distance fitness function. To generate traps with specific characteristics (e.g., coherence), the fitness functions also needed to be correctly aligned with a specific target set. In other words, optimizing for lethality did not reliably produce coherence, and vice versa. Only when we designed a fitness function that was intelligently aligned to a specific goal was the genetic algorithm able to successfully produce traps with either the structural or functional characteristics sought.

ACKNOWLEDGEMENTS

The authors would like to thank Cynthia Hom and Amani Kilaas-Mainas for providing access to their code and helpfully answering questions, and Tim Buchheim for assistance in experimental set-up. This research was supported in part by the National Science Foundation under Grant No. 1950885. Any opinions, findings, or conclusions expressed are the authors' alone, and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Bhattacharya, M., Islam, R., and Mahmood, A. N. (2014). Uncertainty and Evolutionary Optimization: A Novel Approach. In *2014 9th IEEE Conference on Industrial Electronics and Applications*, pages 988–993.

Bock, W. J. and Wahlert, G. V. (1965). Adaptation and the form–function complex. *Evolution*, 19.

Chapman, C. D. (1994). *Structural Topology Optimization via the Genetic Algorithm*. PhD thesis, Massachusetts Institute of Technology.

Fonseca, C. M., Fleming, P. J., et al. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. In *ICGA*, volume 93, pages 416–423.

Gero, J. S. and Kannengiesser, U. (2007). A Function–Behavior–Structure Ontology of Processes. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 21(4):379–391.

Golberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Häggström, O. (2007). Intelligent design and the NFL theorems. *Biology & Philosophy*, 22(2):217–230.

Hom, C., Maina-Kilaas, A., Ginta, K., Lay, C., and Montañez, G. D. (2021). The Gopher's Gambit: Survival Advantages of Artifact-Based Intention Perception. In Rocha, A. P., Steels, L., and van den Herik, H. J., editors, *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART*, pages 205–215. INSTICC, SciTePress.

Jin, Y. and Branke, J. (2005). Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317.

Krink, T., Filipic, B., and Fogel, G. (2004). Noisy optimization problems - a particular challenge for differential evolution? In *Congress on Evolutionary Computation, 2004. (CEC 2004).*, volume 1, pages 332 – 339 Vol.1.

Maina-Kilaas, A., Hom, C., Ginta, K., and Montañez, G. D (2021a). The Predator's Purpose: Intention Perception in Simulated Agent Environments. In *Evolutionary Computation (CEC), 2021 IEEE Congress on*. IEEE.

Maina-Kilaas, A., Montañez, G. D., Hom, C., and Ginta, K. (2021b). The Hero's Dilemma: Survival Advantages of Intention Perception in Virtual Agent Games. In *2021 IEEE Conference on Games (IEEE CoG)*. IEEE.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. MIT press.

Montañez, G. D. (2017). The Famine of Forte: Few Search Problems Greatly Favor Your Algorithm. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 477–482. IEEE.

Montañez, G. D., Hayase, J., Lauw, J., Macias, D., Trikha, A., and Vendemiatti, J. (2019). The Futility of Bias-Free Learning and Search. In *32nd Australasian Joint Conference on Artificial Intelligence*, pages 277–288. Springer.

Reeves, C. and Rowe, J. (2002). *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Kluwer Academic Pub.

Then, T. and Chong, E. (1994). Genetic algorithms in Noisy Environment. In *Proceedings of 1994 9th IEEE International Symposium on Intelligent Control*, pages 225–230.

Wang, S., Wang, M., and Tai, K. (2006). An enhanced genetic algorithm for structural topology optimization. *International Journal for Numerical Methods in Engineering - INT J NUMER METHOD ENG*, 65.

Weibel, E. R. (2000). *Symmorphosis: On Form and Function in Shaping Life*. Harvard University Press.

APPENDIX

A Calculating $P_{kill}(x)$ and P_{max}

We take a trap and find all possible arrows that fire (of which there are at most two). Then, we determine the time, given by t_1, t_2, and position, given by r_1, r_2, that these arrows hit the middle cells. These values indicate possible gopher collision points.

Next, we calculate the position of the gopher, $pos(t)$, at a given time t analytically, which depends on the gopher's skittishness and the time it takes a gopher to eat. The skittishness of a gopher is represented by the gopher's departure from the cell as soon as an arrow is fired (regardless of whether or not the gopher is hit). The eating time t_{eat} for a gopher is defined by the probability vector

$$\vec{p} = [p_1, p_2, p_3, p_4, p_5],$$

where, p_j is the probability that a gopher eats for j frames.

Using t_1, t_2, and t_{eat}, we can determine the position of a gopher at time t. First, we define the time at which the gopher starts to leave, to be

$$t_{leave} = \min(t_1, t_2, t_{eat} + 3).$$

Note that we use $t_{eat} + 3$ to represent the maximum number of frames for the gopher to turn back in the absence of arrows; it takes 3 frames to arrive at the food and t_{eat} frames to eat the food.

Finally, we can define the position of the gopher to be

$$pos(t) = \begin{cases} \min(t, 3) & t \leq t_{leave}, \\ \min(t_{leave}, 3) - (t - t_{leave}) & t > t_{leave}. \end{cases}$$

After defining $pos(t)$, we can see that a gopher is hit if $pos(t_1) = r_1$ or $pos(t_2) = r_2$. Let

$$h_i = \begin{cases} 1 & pos(t_i) = r_i \\ 0 & pos(t_i) \neq r_i. \end{cases}$$

Hence, the probability that a gopher survives arrow i after eating for $t_{eat} = j$ frames is

$$P_{survive,i,j}(x) = 1 - h_i P_{k,i},$$

where $P_{k,i}$ is the probability of arrow i killing the gopher on a successful hit (dependent on the thickness of arrow i).

Furthermore, we define the probability of surviving a trap given that the gopher eats for j frames to be

$$P_{survive,j} = P_{survive,1,j} \cdot P_{survive,2,j},$$

which implies that

$$P_{kill,j} = 1 - P_{survive,j} = 1 - P_{survive,1,j} \cdot P_{survive,2,j}.$$

Notice that the probability of surviving a trap is dependent on the t_1, t_2, r_1, r_2, and t_{eat}, however we exclude these from the expression for the reader's convenience.

Finally, we must take into account t_{eat} since this determines t_{leave}. We do so using the probability distribution, $\vec{p}$, defined above to compute the weighted sum

$$P_{kill} = \sum_{i=1}^{5} P_{kill,i} \cdot p_i.$$

Using this framework, we can see that the maximum probability of a gopher dying is

$$P_{max} = 1 - P_{k,w}^2,$$

where $P_{k,w}$ is probability of a wide arrow killing a gopher on a successful hit.

Code Repository The code used for experiments and visualizations, as well as the data our experiments generated, can be found at (https://github.com/AMISTAD-lab/gopher-grounds-source).