6-2012

# Evolutionary Fabrication: An Autonomous System of Invention

Tim Kuehn
*Union College - Schenectady, NY*

# Evolutionary Fabrication
# A System of Autonomous Invention

By

Tim Kuehn

* * * * * * * *

Submitted in partial fulfillment
of the requirements for
Honors in the Department of Computer Science

UNION COLLEGE

June, 2012

**Abstract**

KUEHN, TIM Evolutionary Fabrication: A system of autonomous invention implementing evolutionary algorithms with rapid prototyping.
Department of Computer Science, June 2012.
ADVISOR: John Rieffel

Evolutionary algorithms have had success in designing complex objects, ranging from antennae used in NASA's Space Technology 5 mission to astronomical telescope lenses. However, evolutionary design is limited by the ability of a simulation to accurately represent the physical world. Additionally, evolved designs may be well described, but they carry no set of specific instructions describing how to physically create such a design. Evolutionary Fabrication (EvoFab) rectifies this: EvoFab is a machine built upon a process that can, in principle, automatically invent and build anything, from soft robots to new toys, by evolving the *process*, not the *product*. We have designed EvoFab, which consists of four components: A) a genotype for printing objects, consisting of a linear set of instructions sent to a Fab@Home, an open-source 3D printer; B) a way to evaluate printed objects using custom machine vision algorithms; C) a way to automate printing by implementing a custom conveyor belt; D) a way of elaborating upon designs by implementing a genetic algorithm. In the near term, we aim to produce an evolved arch. Current results indicate increased fitness over time. Future improvements are possible through restrictions in extrusion along the Y-axis as well as refining fitness evaluation to be less exploitable.

## 0.1 Introduction

In *Star Trek: The Next Generation*, machines called replicators are capable of creating any object asked of them: food, toys, clothing, and spare parts for spacecraft repairs are all available at the push of a button. While such a machine is still far off, we believe that, in some ways, it is possible to do even better: the replicators required knowledge of how to create its products before someone could ask for those products. What if, instead, someone could ask a machine to build something that it had never built before – something of which it doesn't even have any knowledge of how to construct – and it could *evolve* such an object before the person's very eyes? This futuristic vision is the motivation for our research, and it begins with evolutionary algorithms.

What, exactly, is an **evolutionary algorithm** (EA)? EAs are grounded in Charles Darwin's ideas of evolution and survival of the fittest. The basic premises of survival of the fittest and evolution are simple: within a population there exists a certain amount of genetic variance between its members. These variations lead to some members having a slightly higher survival rate than other members of the population; with this increased survival rate comes a higher chance of producing offspring. Those offspring, in turn, will share the same genetic variants and will also have a higher rate of surviving to produce offspring. Over time, newer generations will have variations of their own; when the length of time is significant enough, the characteristics can spread to the entire population and can produce markedly different characteristics when compared to previous generations. It is in this way that animals are theorized to evolve.

One type of EA, called genetic algorithms and described in Algorithm 1, while not supposing to actually present an accurate depiction of Darwin's evolution, are useful tools modeled after the general premises. First, a population containing randomized solutions to a particular problem is initialized. Once a population is initialized, each particular solution is evaluated to see how well they solve the problem – i.e., how "fit" they are. None of these solutions will actually solve the problem by itself, but some may get closer than others. The worst solutions from the population are eliminated, or culled, leaving the better solutions to "breed" a new generation. The chance for any given solution to be chosen as a parent of a new offspring is proportional to its fitness, decided during evaluation. Thus, best fit solutions have a tendency to breed with other high-fitness solutions, producing offspring with the best traits of both parents. In this way, over many generations – tens to hundreds, possibly even thousands of generations, depending on the particular problem – high-fitness solutions can be evolved.

Evolutionary algorithms have had success in designing complex objects, ranging from antennae used in NASA's Space Technology 5 (ST5) mission (Lohn et al., 2005) to astronomical telescope lenses (Al-Sakran et al., 2005). As part of its New Millennium Program, NASA's ST5 mission was "to test, demonstrate and flight-qualify innovative concepts and technologies in the harsh environment of space for application on future space missions" (Lohn et al., 2005). In other words, ST5 mission's intent was to test different technologies that NASA was considering implementing into real space

**Algorithm 1** Genetic algorithms, a form of evolutionary algorithm, find solutions by "breeding" well-fit solutions to combine their best elements.

```
P ← initializePop()
for member ∈ P do
    member.fitness ← evaluate(member)
end for
while P.best.fitness < desiredFitness do
    P ← cull(P)
    newP ← P
    while len(newP) < popSize do
        parent1, parent2 ← chooseParents(P)
        children ← breed(parent1, parent2)
        for child ∈ children do
            child.fitness ← evaluate(child)
        end for
        newP ← newP + children
    end while
    P ← newP
end while
```

missions in the future, to see if they would hold up under environmental conditions in space. The mission involved sending three small satellites, "micro-sats," into Earth's magnetosphere. Each of these satellites required two custom-designed antennae, which happen to be very difficult to hand-design for a variety of reasons, including the difficulty of accounting for complex electromagnetic interactions. However, by evolving the antennae designs using evolutionary algorithms, Lohn et al. (2005) produced, within one month, two antenna designs suitable for the ST5 mission satellites.

Al-Sakran et al. (2005), in turn, used evolutionary algorithms to automatically design an optical lens, completely from scratch. They found that genetic algorithms were capable of making "human-competitive" results – lenses that were as good, if not better, than human-designed lenses. Indeed, one of the evolved lenses actually infringed upon a patent of a preexisting lens; another evolved lens improved upon another existing lens patent.

Through these examples, we can see that evolutionary algorithms can lead to fast, competitive real-world designs of important objects. However, evoultionary algorithms have a tendency to exploit their medium. Thus, if an evolutionary algorithm is run in a simulation that doesn't exactly replicate a real-world scenario, one may find evolved objects that are suitable for the software in which they are tested but incapable of fulfilling their original intention in practice. Sims (1994) found that, when simulating a physical environment, the simulation must be "reasonably accurate"; else, various bugs or rounding errors will certainly occur, leading to undesired solutions that obtain high fitness only through the exploitation of these discrepencies between simulation and the physical world. This problem is called the reality gap: desirable characteristics can be missed and undesirable results can occur when solutions are evolved solely in simulation.

One way to skirt this issue is to evolve solutions in simulation but evaluate their fitness in physical tests. Such a method would prevent any high-fitess traits from evolving if they only have high fitness from exploiting simulations. However, what happens when an evolved design is too difficult to transform into an actual object? This problem is called the **fabrication gap**: evolved designs may be well described, but they often carry no set of specific instructions describing how to get to such an end result (Rieffel, 2006). Consider a picture of a soufflé: it may look tasty, but someone with merely a picture would have a difficult time replicating it, due to the hidden process that led to the end result. In fact, Kavraki et al. (1993) have proven that the problem of assembling a final product from blueprints is NP-Complete. Thus, the task of constructing an object from a relatively simple design may not be that difficult for a person to do, but the ease of construction rapidly falls off as the initial design increases in complexity.

Thompson (1997) presented an elegant solution to the problem of the fabrication gap. Tasked with creating a circuit that could discriminate between 1kHz and 10kHz signals, emphwithout the benefit of a clock, he used a reprogrammable circuitboard to evolve the *actual hardware* rather than a circuit design. This was very interesting because, in simulation, a circuitboard alone would never be able to discriminate between two signals without the aid of a clock. However, Thompson (1997) found that, when running real-world tests rather than simulated results, the circuitboard developed a pattern that exploited electromagnetic interactions to successfully discriminate between the two signals.

Watson et al. (1999) later coined the term "embodied evolution" for an approach similar to that taken by Thompson (1997): they showed how an autonomous population of robots could evolve behaviors that would allow them to successfully compete each other, without the aid of a central computer running a genetic algorithm. The main takeaway for our purposes, however, is not that it was decentralized: it's that instead of evolving blueprints that would then later be converted into real products; the robots themselves were directly evolved. This notion of embodied evolution leads us back to our current research.

Our approach, called **evolutionary fabrication** (EvoFab), bridges the reality and fabrication gaps by combining evolutionary design with a physical fabrication process. By doing so, we ensure that any evolved product already comes with the build instructions, in addition to ensuring that any way in which the evolutionary algorithm exploits the process will be a replicable exploitation. Our machine that implements evolutionary fabrication, called **EvoFab**, works in a three-stage cycle as follows:

1. Print an object
2. Evaluate the printed object
3. Clear the print space for the next object to be printed

This process acts on each member of a population. After a whole population has been printed

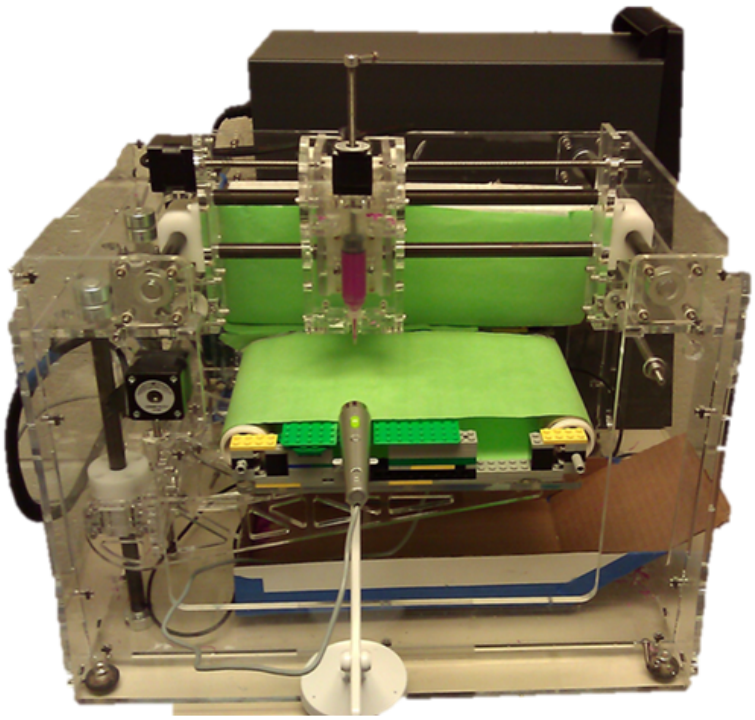

Figure 1: "EvoFab" consists of a Fab@Home printer, computer vision software to determine fitness, and a conveyor belt, all controlled by an evolutionary algorithm.

and evaluated, the best fit members breed a new generation, for which the cycle begins anew. By refining its products over time until it produces a result that fits the design requirements, EvoFab is, in principle, capable of automatically inventing and building anything, from soft robots to new toys.

In this paper, we will detail the previous work, EvoFab 0.1, and will detail the improvements that went into creating EvoFab 0.2. We will describe the individual mechanisms that, together, make EvoFab. We will then show our most currently produced results and discuss current pitfalls of EvoFab 0.2, including ways in which we plan to improve it in the future.

## 0.2 Previous Work: EvoFab 0.1

With EvoFab 0.1, Sayles and Rieffel (2010) created a system that was, at its core, an interactive genetic algorithm. An interactive genetic algorithm is the same as a regular genetic algorithm with one detail: it relies on a human to evaluate each member of the population. Clune and Lipson (2011) have successfully used interactive genetic algorithms to evolve three-dimensional representations of

objects that can be fabricated; however, they are also the first to make note of the fact that evolution in this form is based on subjective measurements made by the human evaluator. When the goal is inherently the pursuit of artistic aesthetics, such as was the case with their research, this does not become a very large problem. However, when more objective measurements are required – as with EvoFab 0.1 – a person's subjectivity can be a large bottleneck in the algorithm's success. For example, Clune and Lipson (2011) found that human evaluations tend to heavily favor symmetry in determining fitness of 3-D models; it can be inferred, then, that attempts to evolve shapes that have inherent asymmetry may prove quite difficult.

Despite the early limitations of EvoFab, Sayles and Rieffel (2010) have shown that it is possible to implement a genetic algorithm into the process of fabrication, successfully using a process of evolutionary fabrication to print two-dimensional objects, such as letters. Such an interactive evolutionary algorithm requires a person to judge with their own, subjective eyes the best products of a generation. Additionally, between generations, a person would be required to physically remove the printed objects and reset the platform to be ready for more objects to be printed. By addressing these two primary issues, EvoFab (Figure 1) is now capable of making its own judgment of the fitness of printed objects, in addition to being able to fully automate the process.

## 0.3 EvoFab 0.2: A Fully Automated Evolutionary Fabrication System

EvoFab is, at its basest, a machine that combines an evolutionary algorithm with a fabrication process. EvoFab creates each population via a three-stage process (Figure 2): first, an object is printed; the object is then evaluated; then, the object is moved off the printing platform to begin the process anew. The information for each printed object is stored within a genotype controlled by a genetic algorithm, written in python. What follows is al deconstruction of EvoFab 0.2 into its basic components.

The primary process of fabrication is controlled by Fab@Home, an open-source 3D printer designed by Malone and Lipson (2007) and useful for its low cost and relative ease of use. Fab@Home operates by extruding material through a syringe and depositing it onto a platform, constructing objects layer-by-layer. The carriage that holds the syringe is free to move along the X- and Y-axes. The platform upon which the material is deposited is free to move along the Z-axis.

Fab@Home is currently in its second revision, Model 2 (Lipton et al., 2009); however, due to currently greater access to the API of the Model 1, we have used the Model 1 as the basis for EvoFab. Additionally, previous work attempted to make use of the Model 2's ability to extrude plastic, which would have allowed for magnitudes longer durations of printing without material refills. However, printing with plastic requires the plastic to be melted, and the associated high temperatures to do so necessitate caution and constant vigilance by the user. Because this negates the ability of EvoFab

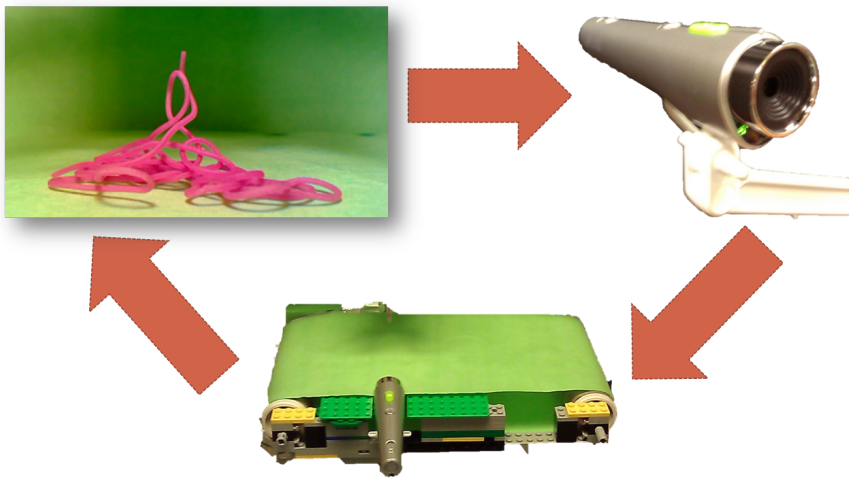

Figure 2: A graphical representation of the three-stage process: print, evaluate, recycle.

to act autonomously, it was decided that the long-term benefits of other materials (discussed below) outweighed the ability of plastic to print for long periods without refills.

Fab@Home normally builds its products by interfacing through USB with a program that contains the products' blueprints in .STL files. However, our Fab@Home is outfitted with a serial connection in lieu of USB, allowing us to send commands directly to Fab@Home (Sayles and Rieffel, 2010). The utility of this is that it allows the evolutionary algorithm to store genotype encodings as a list of commands. The commands that we use to control the printer's actions are as follows:

- **extrude** – This command causes a small amount of material to be deposited onto the print platform.
- **beginExtrude** – This command, rather than send a command directly to the printer, controls the action of the other commands. When activated, all other commands except endExtrude will send their command coupled with an extrude command. Effectively, all other commands say "do this while extruding" when beginExtrude is activated.
- **endExtrude** – This command deactivates beginExtrude.
- **goUp** – Raises the print platform. The platform starts at its max height and cannot be moved upward until a command has been sent to lower it first. This prevents problems of the platform bumping into the syringe (which would, in effect, break the system).
- **goDown** – Lowers the print platform.
- **goLeft** – Causes the print carriage to move left along the X-axis (in the negative X-axis direction). goLeft and its similar commands all act within certain bounds, outside of which

the command ceases to effect movement of the print carriage. For example, the X-axis range may be [-300, 300], so that if the X-axis position is currently -300, goLeft will have no effect on print carriage's movement.

- **goRight** – Causes the print carriage to move in the positive X-axis direction.
- **goIn** – Causes the print carriage to move toward the back of the Fab@Home (along the negative Y-axis direction).
- **goOut** – Causes the print carriage to move in the positive Y-axis direction.

With a printer ready for use, the next step was deciding what material to print with. Previously, Sayles and Rieffel (2010) chose silicone bath caulk as the material of choice. With new goals, however, come new requirements, and after attempts with plastic (described above) and silicone caulk, we settled on a brand of modeling compound similar to Play-Doh. Silicone caulk is easily extrudable, readily available, and comes in many colors, which is useful in allowing computer vision software to easily differentiate a printed object from its background. However, it is also sticky when first printed, and its cure time of approximately thirty minutes for faster-drying variants is too long to wait between prints. Thus, the material would inevitably stick to the print platform, making automation difficult. Some workarounds were attempted (discussed below), but the Play-Doh variant proved much more usable for our purposes. It has the same benefits of being readily available in many colors and easily extrudable without the drawback of stickiness upon first being extruded. This lack of stickiness comes with its own set of problems (see results), but it has proven to be the best option that has been tried thus far.

When printing objects, original iterations did not have movement in the Z-axis due to restrictions put on the system by the camera (see below). However, more recent revisions to the vision system allowed us to free Z-axis movement. This, in turn, allowed us to set the platform much closer to the syringe tip at the beginning of each fabrication. Previously, the platform was at a constant distance of a few inches from the syringe tip. This tended to cause circular extrusions in which the thread of material would spiral downward. This effect, in turn, caused a high degree of unpredictability in how certain instructions would translate to the print: for instance, a command to extrude in a straight line along the X-axis previously would have resulted in something more similar to a sinusoidal function than a linear thread. Now, however, this has been vastly improved, as the average distance between syringe tip and platform is much smaller.

In moving away from the blind watchmaker algorithm, we have devised a new method of evaluation: using openCV wrapped in Python, we have developed computer vision software that works in tandem with a camera affixed to the front of the printing platform. In this way, we can reliably control the method of evaluation. The camera that we used for this process is an Ipevo Point 2 View USB Camera (Figure 3), useful for its ability to focus on close-up images. Additionally, the Point 2 View is supported by open-source Ubuntu drivers, making installation and access to a useful API


Figure 3: A stock photograph of the Ipevo Point 2 View USB Camera: the Point 2 View is useful for its close-up focusing capabilities as well as its ease of positioning. Source: http://www.everythingusb.com/ipevo-p2v-usb-webcam-18262.html

relatively easy. Additionally, the Point 2 View is remarkably versatile in its ability to be positioned in various ways: it comes with a clip that can be attached to virtually anything, allowing us to attach it directly to the print platform. This ability has allowed us to free the platform to move along the Z-axis without worrying that the platform will move out of visible range of the camera.

To allow for automation, we introduced a conveyor belt into EvoFab. Once an object has been evaluated, it is moved off of the platform and deposited in a disposal container via a conveyor belt that interfaces with the evolutionary algorithm via USB. Through this system, EvoFab can run unattended for approximately thirty minutes before requiring a refilled syringe. Not including refilling, EvoFab can run unattended indefinitely. Thus, the only factor that currently inhibits EvoFab from working completely independently of human attendence is, simply, a large enough syringe.

Initially, when we were still testing plastic printing, we used an Automated Build Platform Kit developed by MakerBot Industries for use with their own MakerBot 3D printer (Figure 4). Because the MakerBot also uses plastic, the Automated Build Platform provided a heated base to print onto and would have meshed very well with the printer. However, when switching to silicone caulk, we needed to devise a new conveyor belt, because the Automated Build Platform, as a conveyor

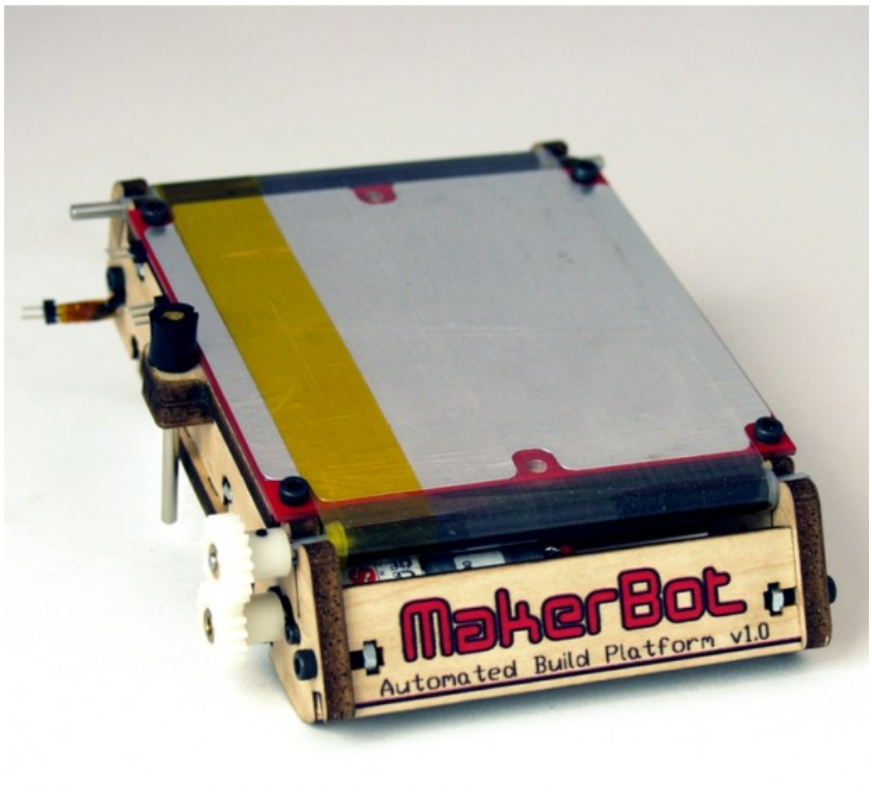


Figure 4: Stock photograph of the MakerBot Automated Build Platform. Originally useful for its heated platform, after further iterations of EvoFab it became obsolete. Source: www.makerbot.com

belt, would have quickly become too messy to be useable. We therefore devised a new conveyor belt that worked much like a scroll works: instead of depositing products after being printed and evaluated, the "scroll" would roll it up into one side of the scroll, destroying the print but clearing the platform. This solved the short term problem of the objects sticking to the platform; however, it merely delayed the inevitable work required of a human to fix the platform. Once the scroll ran out on one end, it needed to be replaced. Thus, it did not entirely solve the problem of complete automation. Thus, when we converted to modeling material, it was a natural transition to turn the "scroll" back into a conveyor belt much like the initial Automated Build Platform. There were two primary reasons why we did not go back to the original MakerBot conveyor belt: first, it was simpler to build a larger conveyor belt, given that the Fab@Home used a much larger platform than the size of the conveyor belt. Second, a custom conveyor belt allowed us to control the color of the surface, making the job of the computer vision software that much easier.

## 0.4 Proof of Concept: Evolution of Arches

In evolutionary algorithms, a different software implementation is required for each type of object one wishes to evolve. Thus, due to the proof-of-concept nature of this research, we implemented software that evaluates exactly one type of object. In choosing what object to evolve, we looked for some shape that is currently not easily produced by a 3D printer. The reason we wanted something that is not easily produceable is that evolutionary algorithms are only truly useful in designing things that have not previously been designed – otherwise, there would be no need to use the evolutionary algorithm in the first place. In determining what a 3D printer cannot easily produce, we focused on the fact that Fab@Home always prints from the bottom-up. In other words, it builds upwards, layer by layer, and cannot construct an object with a portion that is "floating" without any supporting material underneath. Consider, for example, an arch. An arch's supporting columns are easily constructed by Fab@Home, but how will the middle area be created? It cannot deposit material onto mid-air. While a software-limited approach may have trouble desigining a method of construction for such an arch, an evolutionary algorithm is not restricted to follow any set of rules, allowing it to freely explore all possibilities, thought-of or unthought-of by the algorithm's creators. Thus, the potential for finding a solution greatly increases when using an evolutionary algorithm, making "archness" a good object to evolve as a proof of concept.

In evaluating archness of a printed object, fitness increases proportionally to the percent of overhanging mass that an object contains. Figure 5 shows how such a fitness is evaluated: an image is captured by a camera that views the printing stage. Then, the image is thresholded so that the printed object is white and the background is black. This is made simple by printing in a color negatve to that of the background, in this case pink being the negative of green. Then, a bounding box is drawn around the contours of the white image. For all pixels contained within the bounding box, fitness increases for every black pixel that is vertically below a white pixel in its column. Fitness is then divided by total pixels within the bounding box to account for different sized objects, returning the percentage of overhanging mass in the image.

In determining the evolutionary algorithm to use, it was worthwhile to consider the differences between genetic algorithms and random mutation hill climbers. Both genetic algorithms and random mutation hill climbers both create new generations based on the best fit of previous generations, but the way in which they do so is meaningfully different. With genetic algorithms, a child can be created in the following ways:

1. A child can be based off of one mutated parent, in which a combination of the following actions occurs: random instructions in the parent's genotype are deleted, new instructions are randomly added, and existing instructions are mutated into different instructions.

2. A child can be based off of *crossover* between two parents: each parent provides a segment of its genotype, and the two segments are combined to make the genotype of the new child.

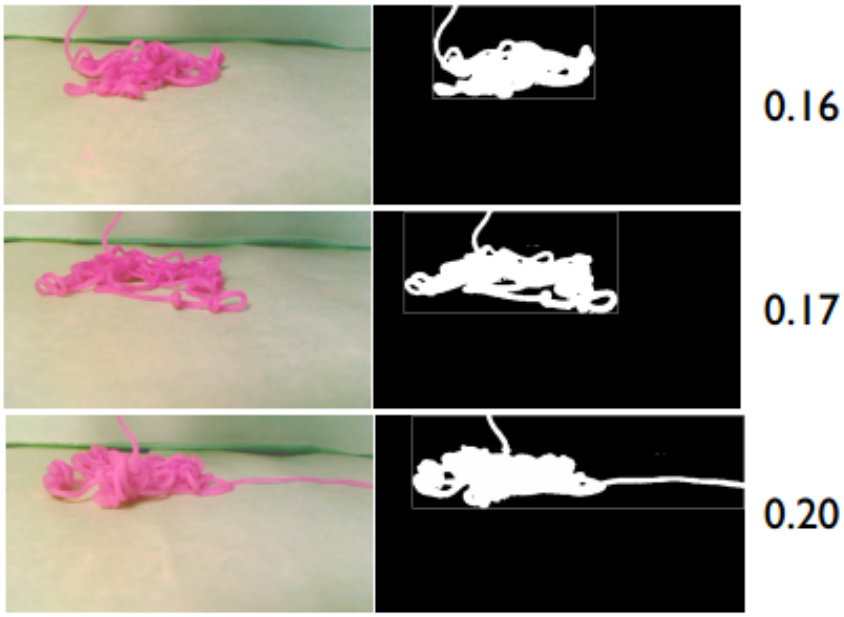


Figure 5: Evaluating "archness": Fitness is determined by first, thresholding the image into black and white, and second, drawing a bounding box around the object and calculating the percentage of overhanging mass.

Compared to genetic algorithms, random mutation hill climbers create children only though method 1. Additionally, the type of hill climber used in this research, called a 1 + N hill climber, differs in another way: each new generation, of size 1 + N, consists of 1 parent and N children all created from that same parent. In other words, all but the one best of each generational population are culled before choosing a parent to use as the basis for the new generation.

Although genetic algorithms may seem more versatile at first glance – and they are, in some ways – they are not, in many scenarios, any more efficient than 1 + N hill climbers. In practice, genetic algorithms are only more useful when solutions to the problem at hand are known to be formed from *building blocks* (Mitchell et al., 1991) that can be joined together, through crossover, to form higher-order solutions. The implemenation differences between the 1 + N hill climber algorithm and the genetic algorithm are as follows:

- With the 1 + N hill climber, **cull**($P$) will always cull all but the single best member of population $P$
- The 1 + N hill climber only chooses one parent when mutating (beceause there is no crossover and thus two are not needed), and because the population of P after culling is always 1, **chooseParent**($P$) always chooses the same parent to mutate.

- Instead of calling the function **breed**(parents), the 1 + N hill climber calls the function **mutate**(parent).

With the sort of problems that EvoFab is currently trying to solve, there is little to suggest that building blocks exist. Thus, for our purposes, a 1 + N hill climber is satisfactory, the pseudocode of which is seen in Algorithm 2.

**Algorithm 2** Random mutation hill climbers are a form of evolutionary algorithm that search for fit solutions by "climbing" the fitness slopes of the search space by making slight random changes to current best-fit solutions.

```
P ← initializePop()
for member ∈ P do
    member.fitness ← evaluate(member)
end for
while P.best.fitness < desiredFitness do
    P ← cull(P)
    newP ← P
    while len(newP) < popSize do
        parent1 ← chooseParent(P)
        children ← mutate(parent1)
        for child ∈ children do
            child.fitness ← evaluate(child)
        end for
        newP ← newP + children
    end while
    P ← newP
end while
```

### 0.4.1 Results

Over sixteen generations of running EvoFab on a 1 + 4 random mutation hill climber, best fitness per generation increased only three times, as seen in Figure 6. However, average fitness makes quite large jumps between low fitness (around 0.17) to high fitness (around 0.26).

In our runs of EvoFab, we have also found the evolutionary algorithm to consistently exploit the fitness function in an undesirable fashion. The fitness function, as it currently stands, captures a three-dimensional image but treats it as if it were two-dimensional. Because of this, pixels that occur at higher elevations in the captured image are treated as if they are higher along the Z-axis; in reality, they may only be further along the negative Y-axis, which the fitness function treats as nonexistant. Thus, archness is frequently measured to be higher in objects that have an "arm" extending forward on the Y-axis, as seen in the second and third images of Figure 5.

Another undesirable exploitation that occurs when printing is that, if the material is not extruded quickly enough, it will not have time to stick to the platform, causing the print carriage to drag around the thread of material instead. This has led to a certain degree of unpredictability in how

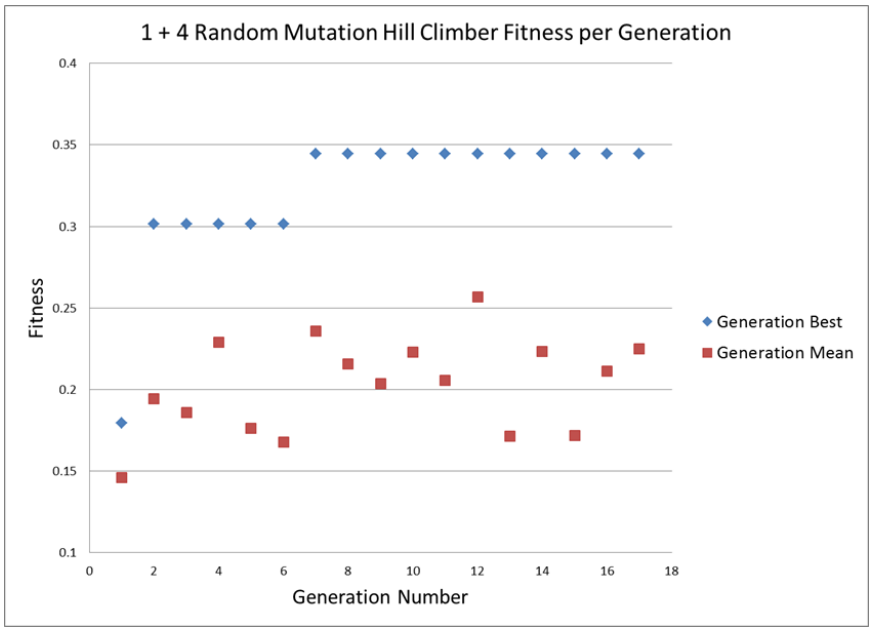


Figure 6: While best fitness per generation rarely increases, average fitness jumps quite drastically for better and for worse. One possible cause is too many degrees of freedom.

a given set of instructions will translate into a printed product. In some cases, this can even cause material that has already been deposited onto the print platform to be dragged along with the hanging thread. This is an example of a mostly-negative exploitation that would go unaccounted for in a simulation-only evolutionary environment. Because of the dramatic effects on overall fitness that this can lead to, we believe that 3-D printers in particular, such as Fab@Home, lend themselves to a method of embodied evolution.

One exploitation that may still turn out to have interesting effects is that, when the platform moves upward along the Z-axis until it is very close to the tip of the syringe, sometimes the syringe tip moves across the object and "picks up" material, moving it to form a new design. This is an example of a potentially positive exploitation that would be nonexistent in a simulation-only evolutionary environment.

## 0.5 Challenges in EvoFab

As seen by the types of evolutions in Figure fig:evaluation, the horizon line of the stage greatly affects measured fitness. When an image is evaluated, pixels higher on the Y-axis are assumed to higher in elevation. However, when the camera captures an image while looking down upon a printed object – thus causing the horizon to shift upward – this measurement becomes skewed. One possible cause

for this is that there are too many degrees of freedom in the movement of the carriage. An arch is well suited to be represented in two dimensions; thickness is no concern. Thus, the carriage's extra variability afforded by movement on the Y-axis may very well be the cause of these jumps. Future tests that restrict movement along the Y-axis – so that only movements up, down, left, and right are allowed – will be able to test this hypothesis.

Another possible way in which this can potentially be fixed in future iterations of EvoFab could involve changing the weighting of the fitness function. Currently, an equal weight is applied to all black pixels found beneath white pixels. If, however, we instead weight more heavily black pixels found at higher elevations in the two-dimensional representation, we speculate that this would offset some of the erroneous fitness encountered below the horizon line. We do not want to completely discount the data below the horizon line, because it could still potentially lead to true higher fitness in non-exploitive ways.

With respect to the dragging of threads that have not yet been deposited onto the print platform: while it may seem like increasing extrusion rates would solve this problem, it is not so simple in practice. The problem does not occur constantly, and it is difficult to predict when it will. It is likely due to an uneven dispersion of pressure throughout the syringe, causing some areas to dispense more quickly than others. There is no easy solution, and it presents a problem in that it will always lead to some unreliability in reproducibility of prints. The best way to control for this is careful preparedness in how material is inserted into the syringe. Avoiding air bubbles and using the same method of insertion every time will help control for these kinds of problems. Again, this is another example of a physical-representation that could not be easily represented in simulation.

Regarding the phenomenon of the syringe tip picking up and moving about material, while no positive results have yet been seen from this, it is likely that this exploitation could potentially lead to objects that would not be able to be formed in simulation alone. One could imagine the syringe tip dragging a a dangling thread of material across two pillars, forming a makeshift arch. Hard to anticipate and almost impossible to control manually, this is exactly the kind of thing that makes software simulation so much harder to use effectively.

## 0.6 Conclusion

We have constructed a system of Evolutionary Fabrication, the first closed-loop cycle of evolving physical objects. The system consists of four components: A) a genotype for printing objects, consisting of a linear set of instructions sent to a Fab@Home, an open-source 3D printer; B) a way to evaluate printed objects using custom machine vision algorithms; C) a way to automate printing by implementing a custom conveyor belt; D) a way of elaborating upon designs by implementing a genetic algorithm. We have shown that the system can produce objects of increasing fitness over time. We have additionally shown how embodied evolution can uniquely exploit physical realities of

various parts of the fabrication process. We have additionally specified areas that currently hamper the system.

Future work will consist of bettering the reliability of EvoFab 0.2 through improvements to the accuracy of its fitness function and restrictions of movement along the Y-axis. Additionally, new types of objects will be presented for evolution to test the versatility of the system, along with new fitness functions to tackle those objects.

# Bibliography


Al-Sakran, S. H., Koza, J. R., and Jones, L. W. (2005). Automated re-invention of a previously patented optical lens system using genetic programming. In Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J. I., and Tomassini, M., editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 25–37, Lausanne, Switzerland. Springer.

Clune, J. and Lipson, H. (2011). Evolving 3d objects with a generative encoding inspired by developmental biology. *SIGEVOlution*, 5(4):2–12.

Kavraki, L. E., Latombe, J.-C., and Wilson, R. H. (1993). On the complexity of assembly partitioning. *Information Processing Letters*, 48(5):229–235.

Lipton, J. I., Cohen, D., Heinz, M., Lobovsky, M., Parad, W., Bernstein, G., Li, T., Quartiere, J., Washington, K., Umaru, A.-A., Masanoff, R., Granstein, J., Whitney, J., and Lipson, H. (2009). Fab@home model 2: Towards ubiquitous personal fabrication devices. In *Solid Freeform Fabrication Symposium*, pages 70–81.

Lohn, J. D., Hornby, G. S., and Linden, D. S. (2005). An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. In O'Reilly, U.-M., Riolo, R. L., Yu, T., and Worzel, B., editors, *Genetic Programming Theory and Practice II*. Kluwer.

Malone, E. and Lipson, H. (2007). Fab@home: The personal desktop fabricator kit. *Rapid Prototyping Journal*, 13(4):245–255.

Mitchell, M., Forrest, S., and Holland, J. H. (1991). The royal road for genetic algorithms: Fitness landscapes and ga performance. In *Proceedings of the First European Conference on Artificial Life*, pages 245–254. MIT Press.

Rieffel, J. (2006). *Evolutionary Fabrication: the co-evolution of form and formation*. PhD thesis, Brandeis University.

Sayles, D. and Rieffel, J. (2010). Evofab: A fully embodied evolutionary fabricator. In Tempesti, G., Tyrrell, A., and Miller, J., editors, *Evolvable Systems: From Biology to Hardware*, volume 6274 of *Lecture Notes in Computer Science*, pages 372–380. Springer Berlin / Heidelberg.

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM Press.

Thompson, A. (1997). An evolved circuit, intrinsic in silicon, entwined with physics. In Higuchi, T., Iwata, M., and Weixin, L., editors, *Proc. 1st Int. Conf. on Evolvable Systems (ICES'96)*, volume 1259 of *LNCS*, pages 390–405. Springer-Verlag.

Watson, R. A., Ficici, S. G., and Pollack, J. B. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *CONGRESS ON EVOLUTIONARY COMPUTATION*, pages 335–342. IEEE Press.