Evofabber: Evolutionary Fabrication

Dave Sayles

June 8, 2010

Abstract

The successful construction of evolved object designs is challenging. Physics simulators are a means to replicate the real world virtually, but phenomena such as the fabrication gap prevent certain optimized blue prints from being constructed in the real world. To circumvent this problem, we propose removing the virtual aspect from object design optimization and evolve objects solely in the physical world. To accomplish this, we introduce Evofabber, a fully embodied evolutionary fabricator, which evolves objects instead of virtual designs.

Introduction

Evolutionary design is the process of optimizing instructions on how to build specific objects. Virtually, this process has been invoked several times, with Funes' work on optimizing lego structures [3] and Sameer's work on evolving lenses [1] to name a few. While both of these trials succeeded in producing solutions to their respective construction problems, the former ran into difficulty when using their evolved blue print. In order to build their evolved structures, Funes was required to infer several steps from the virtually optimized plans, specifically to construct the model on a flat plane then reorient it upon completion [8]. Additionally, Lohn's optimization of satellite antennas [5] had similar problems. The 3-D model produced by Lohn's evolution was used only as a reference, requiring the engineer building the antenna to infer extra steps from the virtually evolved blue print [8].

Both Funes and Lohn's problem is indicative of the existence of the fabrication gap [7]. Evolving virtual blue prints may yield the design of a certain object, but it does not explicitly give instructions on how to build them. On the contrary, the optimized object may actually be unbuildable in the physical world. This, in turn, forces the object's builder to infer certain steps from the evolved design, both increasing the complexity of the instructions and creating the possibility of human error to influence the final product.

The purpose of this project is to discover a way to circumvent the aforementioned fabrication gap. Since virtual objects have been show to fall victim to the fabrication gap (see above), we must find a way to either bridge this gap by evolving our objects differently, or taking a brand new approach to instruction optimization. For this project, the latter was chosen. Instead of evolving objects digitally, we will evolve them in the real world. Using a desktop rapid prototyping machine integrated with en evolutionary algorithm, we will physically evolve building instructions for certain models and run those instructions literally in real time. This allows us to circumvent the fabrication gap by not only evolving objects we can definitely build, but also by creating a set of directions that requires no inference of extra steps.

Background

Fabricators

Figure 1: *An example of a CAD model*

Fabricators are a subset of CNC machines used to create prototypes of various objects. The user first creates a CAD model(Fig. 1) of their object. Then the fabricator uses a special algorithm to creates slices of that model. These slices are printed iteratively, from bottom to top, until the model is complete. Until recently, the idea of using a fabrication machine to prototype multiple models in a short amount of time was an unreachable goal. Fabricators have been historically slow and expensive, with excessive costs in both purchasing the machine itself and building the actual prototypes. Due to these shortcomings, the majority of hobbyists and researchers have been unable to do extensive work with fabricators.

However, with the advent of new technology, less expensive and more

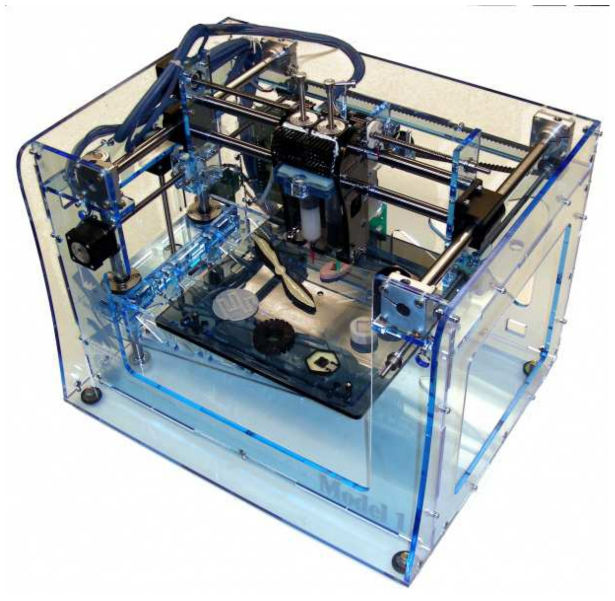


Figure 2: *The fab@home*

conventional fabricators have become available to the general public, for example: the *fab@home*(Fig. 2). The *fab@home* is a relatively small (8"x8"x8" printing space) prototyping machine that fits onto most desktop surfaces. What makes the *fab@home* readily accessible is not its size, though, but its price. Most industrial printers cost anywhere from $20,000 to $250,000; the *fab@home*, at $2,500, is only a fraction of that cost.

The *fab@home* is also quite robust. Due to its use of a syringe to extrude materials onto the printing platform, any material that cures can be used. This means mediums such as: gypsum plaster, room temperature vulcanizing metals (i.e. gallium), plastics, play-doh, frosting, cheese and chocolate are "printable". Due to the *fab@home's* robustness and highly discounted price, it will be the fabricator of choice for this project. While other fabricators offer similar characteristics, the reputability of the fab@home was reason enough to choose it over all others.

Evolutionary Algorithm

Evolutionary Algorithms (EA) incorporate Darwinian principles of evolution to solve problems humans have difficulty completing. Darwin cited the law of *Survival of the Fittest* to explain his theory of evolution. This principle can also be used when solving optimization problems. With an EA, solutions to various optimization problems are paralleled to the different aspects of Darwin's creatures, such as color. Some colors help camouflage into the surrounding environment, making them more desirable than other colors. Creatures with desirable colors will most likely survive longer than creatures with undesirable colors, due to their increased likelihood of evading predators. Eventually, all undesirable characteristics will be weeded out, leaving creatures with the optimal color.

When developing an EA, the user will first compile a list of all the qualities that make up the problem's solution. All qualities are then outlined virtually, creating the genotype of that solution. The next step is determining how to grade each solution's validity. The EA, being an optimization algorithm, needs to know what a "good" solution and what a "bad" solution is. A user defined fitness function gives the EA those details which, in turn, analyzes each solution and gives a specific fitness to its respective phenotype. After all models have been assigned a fitness, a certain portion of the population with the lowest fitness is removed, to make room for new solutions to be created.

To seed the actual population, a random population of solutions is created. When creating new solutions, EAs simulate sexual reproduction by using different genetic operators. In some cases, cross-over will occur; two parents will come together and combine their genetic material to spawn a new child. In others, one solution will be slightly mutated in order to create a new child. In cross-over and mutation, the fitness of each solution is used to determine which solutions will be 'crossed over' or mutated; the better the fitness, the greater the chance of being selected. Both of these techniques aim to birth new solutions which surpass their predecessors in fitness.

The strength of the EA is its ability to thoroughly explore the entirety of the fitness plane (Fig. 3) to find the global optima [6]. In any given problem, there are (usually) several locally optimal solutions, and one globally optimal solution. The local optima are better than their immediate neighbors, but might not be the best possible solution. The global optima, however, is the best overall solution. EA's are incredibly adept at finding this globally optimal solution, due in part to their ability to explore the fitness plane. They do this by keeping an incredibly diverse gene pool, which has solutions from different parts of the fitness plane [10].

Example Evolution

Lets say we want to evolve a set of colors to achieve some optimal color. First, we create our genotype. In this case, our genotype will be standard RGB values for any color. Our fitness function will be the absolute value of the sum total of the difference of all of these values (lets say our optimal color is 50, 120, 210, and the current genotype is at 100, 90, 255. The fitness of that genotype will be: $|(50-100)|+|(90-120)|+|(210-255)|$, or 75. In this case, a fitness of 0 is the best possible fitness). Crossover will choose two relatively fit solutions and swap some of their RGB values, and mutation will randomly deviate some RGB value. Since the genotype is a set of RGB values, the corresponding phenotype would be the actual color. When we execute our algorithm, we seed the population with a certain number of random solutions, use our fitness function, breed new solutions, and keep going until we reach a solution whose fitness is 0. This will leave use with our optimal color, based on our fitness function.

Interactive Evolutionary Algorithms

One specialized type of EA which this project will implement is the Interactive Evolutionary Algorithm (IEA). When assigning fitnesses to certain genotypes, there are situations when a human might want to decide optimal solutions instead of the computer. Imagine an artist evolving an optimal painting, or again trying to evolve colors but not know exactly what kind

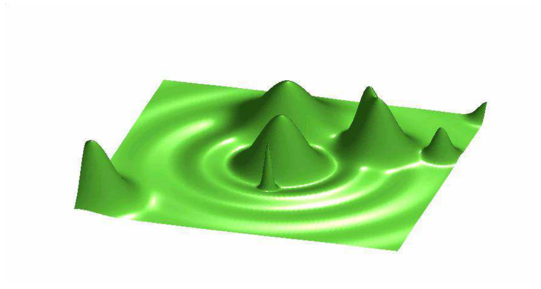


Figure 3: *A Fitness Plane. The higher the peak, the higher the fitness. Not the various high peaks. There is only one highest peak (global optima), yet multiple mid-ranged peaks (local optima).*

you want. Both of these situations would be difficult to encode virtually (its hard to make your desires digital), so having a human act as the fitness function is one way around this challenge. When replacing the fitness function, during the fitness evaluation phase, the algorithm will instead ask the user (person choosing model fitness) to select which models were the most fit. Additionally, because a user is monitoring the algorithm, s/he is given the opportunity to modulate mutation rates, allowing for both more or less mutation to occur, and slightly (low mutation) or radically (high mutation) different solutions to be spawned. Aside from these changes, the IEA runs like a normal EA.

Evofab: An Evolutionary Fabricator

The evolutionary algorithm we have created deviates from your everyday version. Usually, Evolutionary Algorithms are executed in digital space; the genotype and the phenotype being represented virtually. Additionally, because both the genotype and phenotype are digital, the fitness of each solution can be ascertained solely by analyzing bits of data. This setup only works, however, when evolving digital solutions. If one wishes to optimize something physical, several of the components of the Evolutionary Algorithm must be transplanted into the physical world.

Model Genotype

The purpose of the genotype is to represent solutions to the problem we are optimizing. It needs to be precise enough to create potential solution to our problem, yet not so concretely defined that we cannot manipulate it and change some of its parameters. For this problem, we decided to create a genotype with the essential building instructions for models created by a rapid prototyping machine.

When creating genotypes for evolvable models, one approach is to break down a specific model into its primary components; for example a chair would have four legs, a seat, and a back. Each of these components can be assembled in different ways, with different sizes, etc. However, the chair will always have these six parts, and will always be constrained to what we believe a chair to be. This, in a way, squelches the strength of the evolutionary algorithm: the ability to create new and interesting optimizations to problems. Perhaps a chair should be comprised of other parts, or even none of the above.

In essence, a rapid prototyping machine has seven commands: move up, down, left, right, in, out, and extrude. All the commands except for the final command are movement commands; they move the printer head in one of the six listed directions on either the x, y or z axis. The final command, extrude, pushes material out of the printer head and onto the platform. By creating a string of these seven commands, any model that is printable can by printed. Therefore, we do not constrain our models to a small set of "building blocks", but open up our solutions to all possibilities. A linear encoding of these commands is also fairly straightforward to manipulate, making it an easy choice for our genotype.

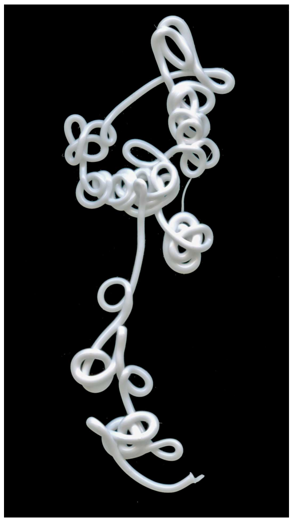


Figure 4: *A sample phenotype.*

Example Genotype: *deedeuioolleeeeoolleuliurllioiieuiiedlueolrureerrddde-drdeuoouierorrorurdudoudelduruooureueeirddrrrlueruoeleiddo*
Each character is representitive of all possible movement commands plus the extrude command: (up, down, left, right, in, out, extrude; each character is the first letter of each command). See Fig 4 for the corresponding phenotype.

For breeding our genotypes, we have chosen the cut-and-splice technique [11]. Unlike regular crossover, the cut-and-splice method of breeding creates a more dynamic gene pool. Random points on two selected genotypes are chosen. All information to the left of the point on the first child is combined with the information to the right of the point chosen on the second child, and all information to the right of the point on the first child is spliced with the information to the left of the point on the second child. Obviously only solutions which do not need all the information encoded in the genotype to create a usable phenotype can use the cut-and-splice operator, since it does carry the potential to loose genetic information. However, it does allow genotypes to grow and shrink during recombination, which can create larger or smaller models. New genotypes are also spawned using older, previously established fit genomes, which can recombine into newer, better models. The cut-and-splice operator was chosen over regular single/multi-point crossover due to its ability to create variable length encodings.

As for mutation, random commands in each genotype are swapped out for a randomly different command (i.e. an Extrude can become an Down).

Phenotype

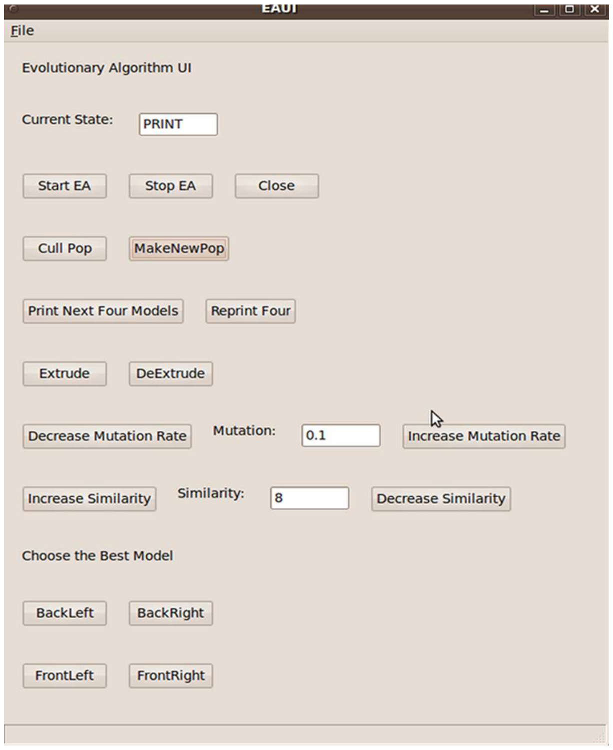
The phenotype is a physical representation of the genotype. It uses the data from the genotype to create a solution which will later be analyzed by a fitness function. Like our algorithm, the phenotype differs from the usual phenotype in that it is a physical, and not a digital, solution. Each model is printed using the string of commands from the genotype. When analyzing fitness for each model, all material extruded is analyzed; any artifacts present will be considered part of the model (see the ”Epigenetic Traits” section).

Fitness Function

One problem that arises from using physical models is how to exactly ascertain the fitness of each model. While it would be excellent to have a system which derives the fitness using image processing techniques and a conveyer belt to move printed models off the platform, it is out of the scope of this project. Instead, we incorporate an IEA to help assign fitness. Models are printed out in sets of four, and the user selects two out of the four models as ”fit”. These fit models are carried over to the next generation, and used to spawn new genotypes. The unfit models are thrown out. Due to this selection scheme, all fit models are, in essence, given a fitness value of 1, while unfit models have a value of 0. This makes operations like fitness based selection and roulette wheel selection obsolete. Instead, we just randomly select the genotypes labeled as fit.

As models become more optimal, the relative difference in fitness between all models decreases. During the initial generations, the more fit models stand out amongst the rest, since they are less refined than their future counterparts. As generations increase, the relative difference in fitness between the models decreases as similarities increase (due to all models beginning to converge on a specific optima), making the selection of fit genomes more difficult. Though this is usually not a problem for a computerized fitness function, due to the human component of the IEA, it creates additional strain on the user [9].

The Algorithm



Current State: PRINT

Mutation: 0.1

Similarity: 8

Figure 5: *The UI used to control both the EA and the printer.*

The algorithm itself is controlled by a UI (Fig. 5), which is a state-machine of sorts. Each phase of the algorithm is paralleled with a specific state: breeding genotypes, culling the population, converting the genotypes into their respective phenotypes and giving feedback of the printed phenotypes. The first two states are relatively starightforward: the breeding state executes the cut-and-splice operation with mutation on the current population to replenish the gene pool. Since, as previously stated, the user chooses half of the population as fit, $PopulationSize/2$ genotypes are always spawned in this state. Culling the population just removes the unfit solutions from the population.

The third state takes four genotypes and converts them to their proper phenotypes. As previously stated, each genotype is a string of commands which can be recognized by a rapid prototyping machine. One genotype at a time, each of these commands is delivered to the fabrication machine through parallel port communication. The four models are printed each in their own designated quadrant I-IV, where each quadrant is a separate corner of the printing platform (back left, back right, front left, front right). While printing, the printer head is bound to whatever quadrant it is in; if it is at the leftmost position in a specific quadrant, any further commands given to the printer head to move left are ignored.

The final state is the feedback state. Embedded into the UI are four buttons labeled ”Front Left”, ”Front Right”, ”Back Left” and ”Back Right”. Each of these buttons corresponds to a quadrant on the fabrication platform. After the print state, with a moded printed in each quadrant, the user can select two of the four buttons which will designate the respective model as fit. After the user selects their desired models, the UI can either go back into the printing state if there are more genotypes to print, or move directly into the cull population phase.

The UI also has functionality which allows for a more diverse gene pool. There are several buttons which either increase or decrease the mutation factor. By increasing the level of mutation, mutated solutions will become more and more different in comparison to their parents. While this does create a more diverse gene pool, it increases the chances that mutated children will be less fit than their parents.

The user also has the option to increase or decrease the similarity between parents and offspring. To avoid duplicates/near duplicates in the population, we use string edit distance to weed out children that are too similar to their parents. Any new children with a edit distance less than a user defined amount will be thrown out to allow for more diverse children to be bred. The pros and cons of this are the same as caveats for the increased/decreased mutation rates.

The evolutionary algorithm we created was developed using Python, with aid from several libraries: PyParallel and WXPython. The former is used for parallel port communication with the fabrication machine. A printer module, which uses a python dictionary to translate commands in the genotype to parallel port signals, was incorporated into the algorithm to aid with printing. The latter library was used to create the UI.

Printer Specifications

Resolution

The concept of resolution is often equated to images, where a certain number of pixels occupy a given amount of space on a screen. Images with greater resolution have large quantities of pixels per square inch, while images with lower resolution have fewer pixels. Printers have different resolutions as well, albeit in a slightly different facet. The image's pixel can be equated to the material extruded out of the printer head; the more beads of material that can fill a given area, the great the printer's resolution. This is also directly related to the size of the stepper motor's "step". By taking smaller steps, the printer is able to extrude more beads onto the platform, increasing the resolution. Conversely, by taking larger steps, the resolution is reduced. Additionally, the size of the printer head that extrudes material also influences resolution size. By having a smaller head, a greater number of smaller dots can be extruded when compared to a larger head.

In an ideal situation, the printer would use the smallest possible step size to gain the greatest resolution. This, however, drastically increases the time it takes to complete one model since more steps must be taken to create a model of substantial size (it is much harder to ascertain the fitness of a small model than a large model). Increasing model construction time would, in turn, increase the overall time taken to complete a successful search. Due to the interactive nature of this algorithm, we must minimize the amount of time it takes to evolve a solution so as to avoid any user fatigue (which may cause the user to select models that are not as fit as others) [9]. This is done by increasing the step size of the motors, and limiting the resolution of the printer and allowing for larger models to be created in shorter amounts of time. In our case, the printer takes 1800 "steps" for each movement command issued. This equates to one full turn of the motor, or 3 mm.

The printer head comes with several detachable cones to extrude out of, each of varying size. Cones with smaller diameters extrude less material and allow for finer resolutions. As stated above, greater resolution yields more time spent on printing, making the smallest cone diameters undesired. Also, cones with large diameters extrude material at excess, creating unwanted artifacts in prints such as sag due to uneven weight distribution and material oozing out of the printer head when it is not extruding. Greater amounts of material extruded also tends to create large pools of material, instead of recognizable shapes. After various test prints, a cone of diameter .8382 mm was chosen, which deposits a stream of material with a width of .8 mm. This allows for models with an acceptable amount of detail to be produces, but not so fine that it takes many extrudes to create something of substance.

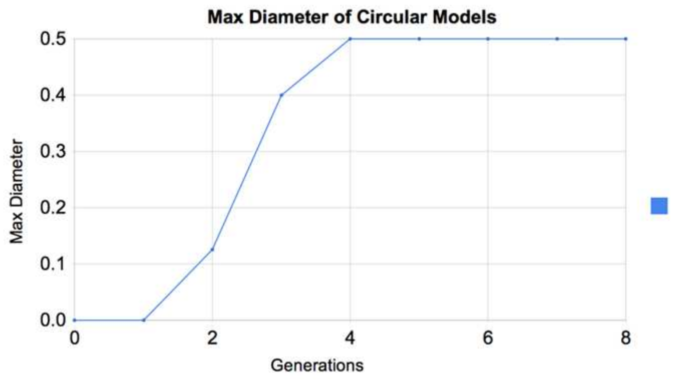


Figure 6: *Max Diameter for evolved circular models.*

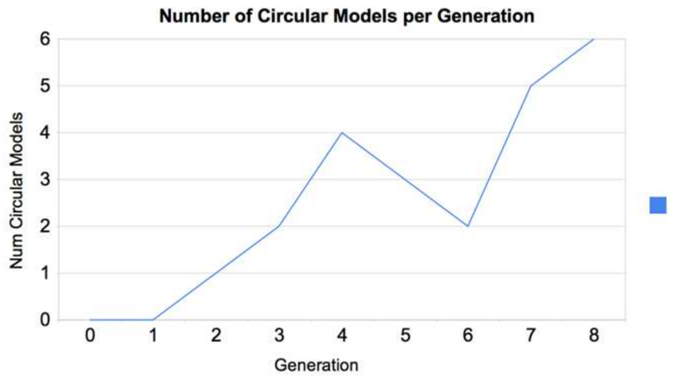


Figure 7: *Total number of circular models present. Note the sudden drop after Gen. 4. This is due to more than 2 circular models being pitted against each other for fitness (i.e. > 2 circular models were printed at the same time).*

A smaller resolution was considered, but upon printing with smaller resolutions it was discovered that printing the same model multiple times yielded different results. This is due to small inaccuracies in both the extrusions of material and the movement of the printing head. When taking small steps, any small deviation in the amount of material extruded, or the way the material falls onto the platform build up to create a large margin of error. By increasing the step size the deviations, though present, are not as apparent.

It is important to note that different materials have different ideal resolutions. For instance, when evolving with silicone, a fixed amount of media was extruded out of the printer head. When evolving with play-doh, we doubled that amount to account for the viscosity of the play-doh (since uncured silicone is much less viscous than play-doh and comes out of the printer head at a much quicker rate as a result). When changing the material, we must always re-calibrate the machine's step size and extrusion size to find the best resolution for our evolution.

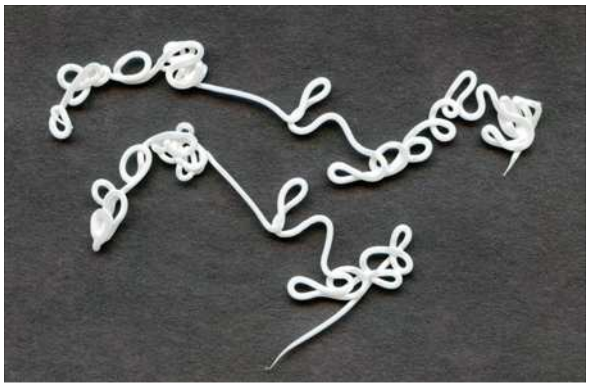


Figure 8: *Two models that both illustrates the breeding of models and proper calibration. The left sides of these models are both (nearly) identical, but the right side is different. It is very possible both of these models had one parent in common, which makes them "brothers".*

Material

Rapid prototyping machines have the benefit of being able to extrude almost any material which is viscous enough to stay in a 10cc syringe and able to cure in a relatively short amount of time. Multiple test prints were done with: play-doh, alginate (a light blue substance made of algae), GE Silicone II and a homemade play-doh like substance. GE Silicone II, however, printed the best. GE Silicone II cures in a short amount of time (roughly 30 minutes-1 hour before solid) and is thick enough to not fall out of the printer head during movement. GE Silicone II does have a negative quality: it cannot support heavy loads while uncured. This became a problem when trying to evolve towers, or the tallest possible structures. This will be addressed in full below.

Platform Dimensions

The printing platform itself is 8" x 8". This is neatly divided into four smaller subsections of 4' x 4'. Having 16' of printing space leaves enough room for models to grow as they evolve and creates a buffer between models, so they do not ooze onto each other.

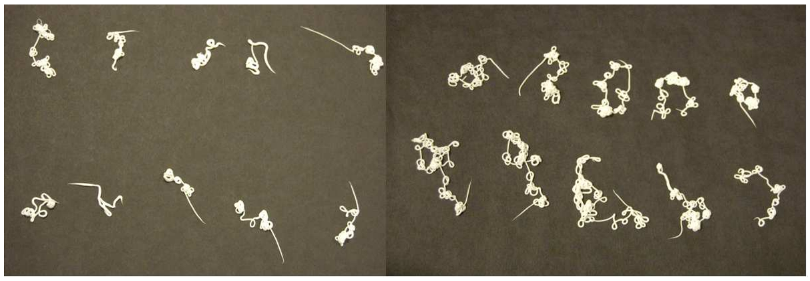


Figure 9: *A side-by-side comparison of circular models. The models on the left are the initial generation, the models on the right comprise the final generation.*

Shape Evolution

The first objects we evolved were simple circular shapes. The initial population was seeded with 20 random genotypes with 20 random commands each. The user selected specific phenotypes as fit depending on their circularity; models that were semi-circular or completely circular were chosen over others. After 9 generations, the population began to converge on circular models, with some expanding beyond that to grow tail-like artifacts (See Fig. 9 for side by side comparison of seed generation to final generation; Fig. 6 and 7 for quantitative analysis).

The second object evolved was the capitol letter A. Again, the population was seeded with 20 random genotypes with 20 random commands each. For fitness, the user selected all models that bore the most resemblance to the capital A.

Around the tenth generation, the population began to converge upon the letter A (see Fig. 10), but had yet to fully complete the model. This is, most likely, due to the lack of genetic diversity in the population due to the absence of mutation. The cut-and-splice operator is usually employed without mutation in order to exploit the initial diversity in the gene pool [4]. However, because the gene pool in an IEA is smaller than a regular EA (and less diverse as a result), the cut-and-splice operator must be paired with genetic mutation in order to maintain acceptable genetic diversity. For all subsequent evolutions, mutation was included. It is of the opinion of the user that had either the evolution continued for several more generations, or a second trial was undertaken to again evolve the capitol letter A with mutation, the goal would have been reached. However, due to time constraints and the need to evolve varied models (due to pressure to fulfill the requirements for the art show), we stopped at the models seen in Fig. 10.

The final models evolved were towers. In this case, the absolute height of a phenotype was the sole decider in fitness. This object was first evolved using GE Silicone II, which was used to evolve all prior models. However, after 10 generations, models were unable to gain a significant amount of height. This is due to the current methodology of the Evofabber. Speed has always been the sole attribute we have stressed; pumping out models quickly leads to more generations, which will yield a more optimized solution. Fatigue of the user is also an important factor; making the user constantly watch the printer for extended periods of time could result in inaccurate fitness selections [9]. However, this paradigm has proved to be double-edged. Because the base level of all models was not allowed ample time to cure, the top levels were never structurally sound, causing them to topple over. This is believed to be the case, since the user witnessed several relatively tall models being printed only to fall over seconds after completion.

During the second round of evolving towers, a different building material was chosen. GE Silicone II is rather heavy, and when uncured does not support much weight. To circumvent this, we tried play-doh. However, play-doh was too viscous, and moving the printer head to succeeding quadrants after printing one model caused that model to be dragged along the platform (or completely unwind to a long string in some cases!). Due to this, a medium which was not as thick as play-doh, yet thick enough to support some weight, was created from flour, salt, oil, and water. Though this material was more successful than GE Silicone II, it was unable to create models with height greater than 10 mm.

From the above trails, it can be concluded material properties need to be examined in full to determine the best possible material for a job. Materials that are not load bearing until cured will be unable to yield more complex structures, since they will just topple over and never gain significant height. However, because we always emphasize speed (otherwise the IEA will take literally weeks to evolve globally optimal models), waiting for objects to cure is not the best course of action. Instead, choosing materials which cure under specific types of light, such as UV light, may be the best idea. These materials will cure in a relatively short amount of time once extruded, allowing them to support more weight than most other materials. UV curing materials are, however, very expensive.

Epigenetic Traits

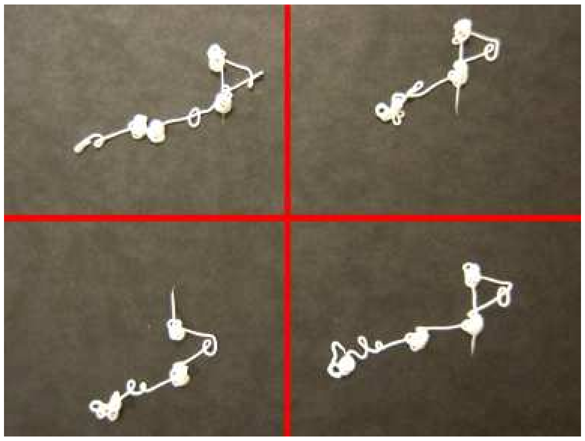


Figure 10: *The above models illustrate the appearance of epigenetic traits as well as the evolved capitol letter A.*

One interesting artifact present in many of A models evolved were traits present in the phenotype but not coded into the genotype. These epigenetic traits often manifested themselves in the form of the cross in the capitol letter A (Fig. 10). After a model is completed, the printer head will move to the center of the next quadrant to start printing again. When this happens, material that was not placed onto the printing platform or material that continued to ooze out of the syringe is dragged across already printed models, creating epigenetic traits. In the case of the A, these traits are only present in models printed in the back left, back right and front right quadrants. This is due again to the actions taken by the printer head, since no extra movements are executed in the front left quadrant.

It is important to note that other epigenetic traits may exist, but they are less evident. The cross of the A is very apparent due to both its location and it being one of the defining characteristics of the letter A.

Uses

The Evofabber has been shown to have the ability to evolve objects. However, not all objects should be evolved using this equipment. Only objects which have properties that are difficult to simulate virtually should be evolved, due to the amount of time and work it takes to use the Evofabber. One type of object which is difficult to model virtually that has a potential with the Evofabber is Soft Robots [8]. By using the Evofabber instead of computer simulations to develop soft robots, the challenge of trial and error can be circumvented, and robots can be fully evolved physically. This does, however, open the door to more complex problems, such as: How will I test the fitness of each model? Ideas such as perturbing the printed models to see the distance moved should be examined.

Future Work

Automation

The one constraint on the Evofab process is time. Though printing the abstract shapes listed above are rather quick (10 minutes for four models, an hour for a generation), printing more elaborate models will take much longer due to the increased resolution and height they would require. To have a user sit at a console just to give feedback for all prints would be too costly to be feasible. By automating the Evofabber, the need for interaction between the user and the algorithm would be removed and it could run autonomously until it produces a globally optimal model.

Such a machine would require us to severely modify the current Evofabber. First, a new fitness function must be created which requires no input from a user. The computer must be able to see all models from different angles, and be able to analyze each one and assign an appropriate fitness. For instance, if we were evolving height, the amount of vertical pixels the model covers could be directly related to its fitness. This adds an additional level of complexity, however, and could be an entire senior project in itself.

Another factor that needs to be put into consideration is how to keep an ample supply of building material. Currently, the volume of material held in a syringe is good for about 12 models **of the size printed in the aforementioned evolutions**. Models that are more intricate or larger will increase the amount of material used, and decrease the amount of models able to be printed from one syringe. If we wish to fully automate the Evofabber, we must incorporate some large resevoir of material the printer can use, that can be easily refilled *while printing* (as to allow refilling of medium while printing, and not require us to either pause or stop printing just to refuel the Evofabber). This would only require sporadic visits from a user to replenish the printer's supply of material.

Creating a completely automated Evofabber would yield benefits other than a more thorough search of the fitness plain; it would also allow for more concrete fitnesses to be assigned. Some differences in phenotypes are so subtle, humans have extreme difficult picking them out (consider again evolving towers, where one tower is 15 mm tall, while another is 14.5 mm tall). Using a computer to do this calculation would remove human error from fitness calculations, and allow for only the best models to pass on to the next generation.

Materials

The materials used in this experiment were Silicone II and a play-doh like substance. Though both of these materials were adequate for 2-dimentional models, creating models with more depth creates a problem. Since silicone does not cure fast enough, it is unable to immediately bear any load since it must first take time (roughly 30 minutes to an hour) to set. For the other substance used (homemade play-doh like material), it unable to bear the load required to gain significant height.

Another problem that arises is the quantity of material required. Constant printing requires constant use of materials, a resource which can become rather price. A material which can be infinitely recycled would be the ideal, since it would bring down the total cost (in dollars) of evolution and allow for constant printing without the need for manually depositing more material in a printing reservoir (assuming it is possible to fully automate the Evofabber process and harvest used materials).

The seemingly ideal material is ice. A *fab@home* that prints ice has already been created with several modifications [2]. Ice would be cost effective, since it is relatively inexpensive. It is also very recyclable. If we were to sweep all models post-printing from the printing platform, we could deposit them into a heated container, which melts them down and puts them back into the main printing reservoir.

An additional problem when considering building materials is the differences between mediums. If, for instance, ice was used to evolve a set of instructions, would those instructions be valid for other materials, such as silicone? Or would the set of optimized instructions need to be modified on per-material bases? More research into the ways materials are extruded out of the Evofabber is required.

Although ice may be ideal for some objects (rigid non-moving objects for example), they would not be ideal for soft robots. Ice does not support fluid motion, and is nowhere near soft. Also, it would not be able to survive in temperatures greater than 0 centigrade. A different material would need to be found for soft robotic optimization.

Conclusion

The Evofabber was able to successfully evolve construction commands for simple abstract shapes. Though no complex models were evolved, the potential exists. However, significant work on creating a robust and reliable automation or the Evofabber must first be done before anything intricate can be evolved.

References

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

[2] E. Barnett, J. Angeles, D. Pasini, and P. Sijpkes. Robot-assisted rapid prototyping for ice structures. In *IEEE Int. Conf. on Robotic and Automation*, 2009.

[3] P. Funes and J. B. Pollack. Evolutionary body building: Adaptive physical designs for robots. *Artificial Life*, 4(4):337–357, 1998.

[4] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In M. Kaufmann, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, 1993.

[5] J. D. Lohn, G. S. Hornby, and D. S. Linden. An evolved antenna for deployment on nasa's space technology 5 mission. *Genetic Programming Theory Practice 2004 Workship (GPTP-2004)*, May 2004.

[6] M. Mitchell, S. Forrest, and J.H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In F.J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1992. MIT Press.

[7] J. Rieffel and J. Pollack. Crossing the fabrication gap: Evolving assembly plans to build 3-d objects. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 529–536, Edinburgh, Scotland, December 2005.

[8] J. Rieffel and D. Sayles. Evofab; a fully embodied evolutionary fabricator. In *International Conference on Evolvable Systems*, 2010.

[9] H. Takagi. Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evalutaion. *Proceedings of the IEEE*, 89(9):1275–1296, Sept. 2001.

[10] R. K. Ursem. *Parallel Problem Solving from Nature - PPSN VII*, volume 2439/2002 of *Lecture Notes in Computer Science*, chapter Diversity-Guided Evolutionary Algorithms, pages 462–471. Springer Berlin, Heidelberg, 2002.

[11] D. Whitley, J.R. Bevridge, C. Buerra-Salcedo, and C. Graves. Messy genetic algorithms for subset feature selection. In *International Conference on Genetic Algorithms, ICGA-97*, 1997.