Genetic Crowd - Crowd-sourced Evolution

Alexander Braylan, Kaivan Wadia, Mark Gray
University of Texas at Austin
Gates Dell Complex 2317 Speedway
Austin, TX 78712

Abstract

Genetic algorithms have been used in many different areas of computer science. Many of these areas focus on using a genetic algorithm to optimize objective fitness functions. We investigate the use of genetic algorithms to optimize an output such as a game or animation along a subjective fitness function. We explore the use of human computation to crowd-source the evaluation of the outputs of such a genetic algorithm to evolve the generated artifact.

Introduction

Genetic algorithms have been used in many different research areas of computer science. These types of algorithms are generally used in optimization and search problems. Genetic algorithms are a sub-section of the larger class of evolutionary algorithms. Genetic algorithms have traditionally been inspired by natural evolution concepts such as inheritance, mutation, selection and crossover. Most common genetic algorithms are initialized with a random population of possible solutions, called candidates, for the problem being solved. Each candidate is defined by a specific set of properties called chromosomes or genotype and is usually represented as a binary string of 0's and 1's. The genetic algorithm progresses by evolving the candidates towards better solutions. The evolution is done by mutating the candidates by changing some or all of their properties to generate new candidates that might be better solutions to the problem. Initially a population of randomly generated candidates is used as the first batch of solutions and the algorithm progresses iteratively with each subsequent batch of solutions, called generations, being evolved from the previous generation. The best candidates of every generation are determined by using a fitness function. Fitness functions are objective functions that are used to evaluate how close a given candidate is to achieving the aims of the problem being solved.

Traditionally, fitness functions involve running the solution against some test cases and evaluating the performance of a possible solution. This is good in cases where the problem is clearly defined objectively and one has lots of test cases to run the solution against. Certain problems do not necessarily have objective solutions, such as whether a game is fun or not or whether a picture is beautiful or not. The answers to these differ based on the person viewing the picture or playing the game. For such problems it is difficult to define a fitness function without considering human input, and even then it is difficult to incorporate subjective input into the fitness function. In this paper we present a system by which we can incorporate feedback from a crowd of human workers into the fitness function in order to evaluate the fitness of a candidate.

Data collection and processing have historically been difficult problems, but with the emergence of crowdsourcing over the last decade these problems have become much easier to solve. Using the power of the crowd one can collect or annotate a large amount of data at a fraction of the original cost and in a shorter period of time. Amazon Mechanical Turk (AMT) has spearheaded this revolution and is currently one of the largest platforms for crowdsourcing. In our system we crowdsource the task of evaluating the candidates of the genetic algorithm through AMT.

This paper aims to answer three main questions about the ability to use human computation to evaluate the fitness of a genetic algorithm's output. What are some ways to guide the genetic algorithm using human computation? How can we use the input from human workers as a fitness measure to evaluate the genetic algorithm's objective output. Finally, does the fitness of the population improve over time even though we give no guidance to the workers on how an output should be evaluated? In other words, do workers agree on the aesthetic quality of novel, arbitrary artifacts? To answer each of these questions, we create a genetic algorithm to produce a subjectively evaluated output, and we use the human computation platform Amazon Mechanical Turk to have the crowd evaluate the fitness of the output. We describe the types of information we collect from the crowd workers and examine their relationships. Finally, we analyze the effectiveness of the genetic algorithm in evolving subjective artifacts and assess the degree of agreement among workers. The next section highlights the previous work in this area followed by details of our system implementation. We later describe the experiment we conducted and the results of the experiment and finally conclude with future directions of our system.

Previous Work

Picbreeder (Secretan et al. 2008) is an example of a collaborative network of users creating and evolving shared images. The users determine the subjective fitness of an image by evolving it until they like the produced image or by publishing it so that other people can view it and evolve it as well. This allows the fitness of an image to be determined based on the user's perspective of an attractive image and not a traditional fitness function determined by the score of the produced artifact. This methodology allows one to determine the fitness of artifacts that are not easily calculated. Sakamoto et al. talk about the different values of a creative artifact, for instance novel and useful or original and valuable. They also talk about the use of crowd workers to be the fitness function for creative work (Sakamoto, Nickerson, and Bao 2011). We aim to implement a similar evaluation process that will be discussed later in the paper.

Endless Forms (Clune and Lipson 2011) is a variation on Picbreeder that uses the same idea of using a genetic algorithm to generate an evolved form of a 3D image and present it to a user. The user then decides whether or not to publish the image or continue to evolve it. Endless Forms also allows the tagging of artifacts based on what they represent. This allows people to classify the images they evolved. Similarly, we are using human computation to determine the subjective fitness of our genetic algorithm's artifact. However, we will not be giving individuals the ability to evolve the artifact within the system, but rather aggregating the crowd's feedback to perform a more directed evolution.

Automatic game generation is a complex task, and requires a subjective fitness function as opposed to traditional quantitative ones. Togelius and Schmidhuber were able to develop a genetic algorithm that developed games based on an objective fitness measure (Togelius and Schmidhuber 2008). Our aim is to use the crowd to create a more useful fitness function. By guiding the genetic algorithm with a fitness function developed by humans we hope to see the overall fitness, from a subjective standpoint, increase as more people participate in the evaluation process.

The next issue we will face is developing a good survey that will capture the necessary information but will not be cumbersome to the user. We focus on developing tasks based on the best practices for survey information described by Alonso (Alonso 2009). These practices are described later in the paper in the Front-End section.

To determine the success of this portion of our project we will need to analyze the input we gather from the crowd. This will be done by analyzing the trend of data and ensuring that there are control questions in place that keep spammers from submitting information. We will also do self-verification of highly clustered artifacts to see what the crowd is determining as attractive. These processes will be covered in more detail in later sections.

System

Our system constitutes three major parts: the algorithm to generate the Interactive Animated Graphic (IAG) for evaluation by the crowd, the Front-end website presented to the human worker displaying the IAG for evaluation, and the Back-end server which hosts all the data and performs the mutation once all the IAGs of a single generation are evaluated. A worker on AMT is redirected to the website hosted by us and is presented with a random IAG of the latest generation for evaluation. Upon completing the evaluation, the worker receives a confirmation code in order to receive payment. Finally, once all the candidates of a generation are evaluated, the mutation algorithm is executed to generate the next batch of candidates. We shall present the Algorithms used followed by a description of the front-end and back-end implementations.

Algorithms

The genetic algorithm produces IAGs which are then evaluated by the crowd. An IAG is specified by a collection of rules, for example a computer program. In order for a genetic algorithm to produce IAGs, each IAG must be encoded as a genome manipulated by the genetic algorithm and interpreted as the defining rules. The specification of this genome determines the potential diversity and constraints in the set of possible IAGs. We currently have one working genome specification and the genetic algorithm to manage it.

IAG Genome The IAGs are inspired by grid-like Atari games. In Atari games, there are a few different object classes defined by movement and interaction rules. Most of the time, the rules acting on an object take input from the grid cells immediately neighboring the location of the object. In (Mnih et al. 2013), Atari agent behavior is learned by training convolution networks which process the input grid in patches of neighboring cells. We also assume that the kinds of rules that will produce the most Atari-like games will be functions of neighboring cell patches. Therefore, the main ingredient in our IAG genome specification is a collection of perceptrons (Rosenblatt 1958) determining the activation of a cell dependent on the input from its neighboring cells. The weights of these perceptrons are defined in the genome, and the total number of weights is the number of neighbors (4) times the number of perceptrons.

The number of specified perceptrons is predetermined to be at most 9, corresponding to regions of the grid space: northwest, north, northeast, west, center, east, southwest, south, and southeast, times a specified number of allowed object classes. Cells in each region of the grid in the same object class all follow the same perceptron rule. This is due to the observation that in Atari games, similar objects tend to occupy similar regions of the grid. The demarcation of the different regions is another set of four parameters defined in the genome.

Having specified the object classes as perceptron rules and the object locations as grid regions, the final major element of the genome is the level design, or the initial activation of the cells in the IAG grid. Two different levels of a game with the same set of rules may feel very different to human players. Therefore, the initial state of the IAG must be encoded by the genome to avoid losing this important information when instantiating a game from a genome.

A few other global parameters are also specified in the genome. One is the granularity, or size, of the IAG grid. Another is the wrapping rule for each perceptron: that is, whether or not a cell on the edge of the grid takes as a neighbor the cell on the exact opposite side. Finally there is the number of perceptrons used as object class definitions, which we allow to vary between 5 and 9.

One important element of games that is missing from the IAG objects is the player agent definition. In Atari-like games, such as the ones generated by the Video Game Description Language (Schaul 2013), there are many different kinds of player agents: ones that move in all four directions, ones that move in two directions but can jump, ones that shoot, ones that melee, etc. In our framework that defines IAGs as perceptron rules on the grid cells, it is currently impossible to incorporate such a player due to the necessary nonlinearity in the function on neighboring cells and action inputs. This difficulty makes IAGs more like animations than actual games. To make them more game-like, we allow human players to interact with them by clicking on cells to toggle their activation. In exploratory testing we found that, even without clearly defined objectives, humans can set their own objectives and enjoy interacting with IAGs with enthusiasm ranging between mild and substantial.

Genetic Algorithm The function of the genetic algorithm is to manage and act upon a population of genome instances, improving the measured fitness of its top-ranking members over generations. In our project, an IAG's fitness is a subjective evaluation provided directly by the player through a survey question.

The main steps in each generation of our genetic algorithm are *grade*, *kill*, and *repopulate*. After grades are produced for each member by averaging its evaluations, the lowest-graded members of the population are removed according to the kill rule. Finally, new members are added according to the re-population rule, which may apply mutations to surviving members, produce cross-bred children of surviving members, or create new randomly initialized members from scratch.

Front-End Website

We developed a website that allowed a crowd worker to interact with our IAGs. The design of the website was made to be simple, aesthetically pleasing, and fast. We wanted the user to have minimal barriers preventing them from completing an evaluation. The Bootstrap framework was used to style the website and make the overall user experience pleasant. Along with Bootstrap, we added custom JavaScript and jQuery code to the website to give the user responsive feedback and a browser agnostic display. The front-end of our project needed to have a simple way of gathering responses from users and sending the evaluation information back to our server. We followed many of the recommendations mentioned by Alonso (Alonso 2009) about how to produce an effective survey design. We self-contained all the information needed on one page and minimized the clutter by hiding non-essential data from the user's view. We maintained a clear set of instructions by stating that the user can interact with the IAG by clicking within it and seeing how it reacts. After the user has lost interest in the current IAG they can click the finish button and fill out a short one (1) question survey and hit submit. Once completed we then send the information from the user's browser to the server which in turn stores the evaluation. The functionality of the server is covered in a later section of the paper.

Gathered Information When an evaluation is submitted there are several details that we collect from the user's session and store on the servers. Some of the information we collect passively without the user being affected while interacting with the IAG, and some feedback is explicitly asked for from the worker. This information is essential to determine the fitness of an IAG and perform the evolution process. Using the gathered information described below we hope that continued evolution increases the overall fitness of the top members of each population.

1. Evaluation score: This is the rating given by the user corresponding to a specific IAG instance. The scale for this field is between 1 and 3. This is the main subjective question posed directly to the user. A crowdworker would have to at least answer this question in order to get paid for the task.
2. Number of clicks: This is the number of times the user clicked within the IAG. This metric may represent how interactive an IAG appeared to be to the user. More clicks may imply a more interesting IAG or user experience.
3. Location and time of clicks: We record all click events that happen within the IAG. This metric allows us to see the clicking pattern during an evaluation, and helps determine if a IAG was responsive to the worker's clicks. We also use this information to measure the total time spent on an evaluation.
4. Member Information: To properly evaluate the members of a generation we need to send the member's information that goes with each evaluation. This is a lightweight representation of the evaluated member and is stored on the server for later use.

Integration with AMT We use AMT in order to acquire the necessary human workers required to evaluate the candidates of each generation. For integration with AMT we will redirect the user to the evaluation page on our website from the Human Intelligence Task (HIT) page. There, a Turker will participate by interacting with the IAG and submitting a one question survey once finished. Upon completion the Turker will be shown a loading screen while the information is being processed. Once the evaluation is stored on the server the worker is presented with a confirmation code for his evaluation which must be entered into a field on the AMT HIT's page. This confirmation code allows us to verify the work done by the worker and pay the worker for the completed task. To prevent spammers we only allow a code to be used only once and keep a record of previously submitted confirmation codes. AMT integration allows us to seed our website with several initial generations in hope of attracting public participants.

Public Participation To avoid only receiving information from one demographic we decided to use more than just AMT for our subjective fitness function. We created a publicly hosted website that allows anyone to freely contribute. Using both voluntary and compensated participation is a good way to gather a large user base. The current version of the website can be viewed at http://geneticcrowd.appspot.com/. We decided not to implement an account system to maintain a simple and fast user experience.

Back-End API

In order to enable crowdworkers to evaluate IAGs and then perform the mutation between generations of the population we had to develop and host a website. We decided to use Google's AppEngine Platform (Ciurana 2009) to develop and host the website. The AppEngine service is widely described as a Platform as a Service (PaaS). This basically means that a customer of this Platform can develop and build a web application and deploy it to the platform which in turn provides the network, servers, storage and other services to host the consumer's application. This is very helpful to a customer as it abstracts away the low level details of hosting a website, such as server management in relation to failures and crashes, and enables the consumer to concentrate on developing features for their web application.

In order to host a web application on Google AppEngine one can do so by either using Python or Java as the programming language of choice. We decided to use Python due to its ease of use and ability to setup quickly. While Java is generally considered a more stable choice for developing an extensive system, we preferred the flexibility provided by Python especially since it is better suited to rapid iteration based development of a small system.

REST API The back-end API on the server had to provide two major functionalities. The first functionality is the ability to store candidates of a population and their evaluations collected from the human crowd workers. The web server should also be able to provide an Application Programming Interface (API) to enable the fetching and storing of these genome instances for each generation. The second task that the web server needs to perform is to detect when all genome instances of a generation have been evaluated and start the mutation algorithm to generate a new generation for the next evaluation round.

We implemented a REST API to store and fetch genome instances from the web server. This involved initially designing a database schema to store each genome instance and link it to a specific generation. We created a model to describe a single candidate using the NDB API provided with Google's AppEngine service. The NDB API provides persistent storage capabilities in a schemaless object store. We also store the evaluation of each candidate of a generation. We collected a minimum of four evaluations for each candidate from the crowd of human workers. These evaluations are stored as part of the candidate model described earlier. We decided against describing and maintaining a separate evaluation model so as to have fewer object models in the long run.

In order to determine whether a given candidate has been evaluated the required number of times, we maintain the state of each genome instance. Initially a candidate of the population is in state *not-evaluated* when it is created. When a crowdworker starts evaluating a particular genome instance it transitions to the state *locked* denoting that this genome instance should not be presented to any other crowdworker. If the crowdworker abandons the evaluation task midway the genome instance transitions back to *not-evaluated*; otherwise upon the completion of the evaluation task it transitions back to the state *not-evaluated*. Upon the submission of the fourth evaluation of the candidate by a crowdworker the state of the candidate is changed to *evaluated*. When a candidate is in state *evaluated* it is no longer presented to a human worker for evaluation.

Mutation Once all the genome instances of a population are in the state *evaluated*, i.e all the candidates have been evaluated at least four times, we execute the mutation algorithm to generate the next generation of candidates for evaluation. In order to detect when to run the mutation algorithm we check the states of all the genome instances every time a crowdworker completes an evaluation of an IAG and it is submitted to the server. We had to decide whether to perform this check every time an evaluation was submitted or at regular intervals through the execution of a *cron* job. We eventually decided on performing the check every time an evaluation was submitted so as to minimize the time spent between generations. If we had implemented this check as a cron job there could have been times when we had a number of human workers ready to perform the evaluation task but no candidates to evaluate, as the cron job's next scheduled execution could be a couple of hours away.

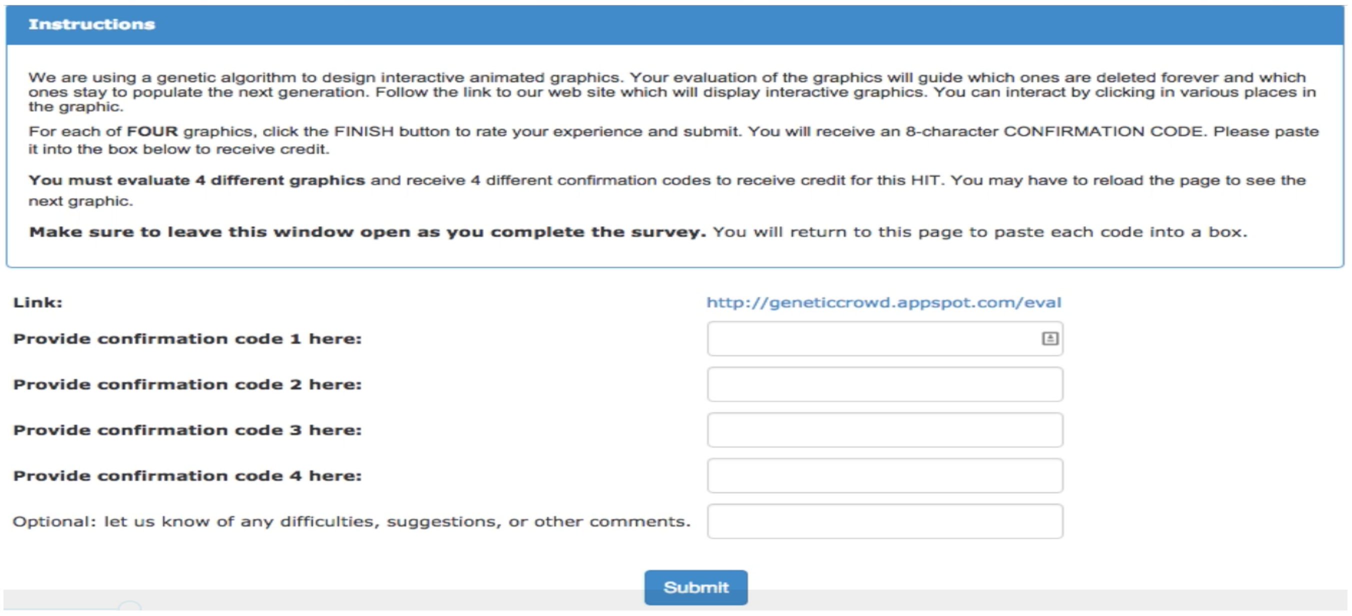
For the purposes of the mutation step we need to determine the fitness of each candidate in the current generation. We compute the fitness of a candidate by calculating the average evaluation score of each candidate based on the scores submitted by the crowd workers on AMT. We currently do not use the information gathered regarding the clicks and the time spent by each user interacting with the game while evaluating it to drive evolution. In the future we would like to make the fitness function multimodal and incorporate this data into the calculation of the fitness of a candidate.

Experiment

We conducted an experiment wherein we used AMT to collect a crowd to evaluate our artifacts generated using the genetic algorithm mentioned in the previous section. We shall now describe the setup and execution of the experiment starting with the initial setup of the generations and then present the results in the following section.

Generation Setup

We setup each generation of the iterative process to contain twenty candidates. The first generation was populated by randomly generated candidates. We determined that twenty candidates per generation would be enough to give us good coverage without becoming unmanageable. Each candidate



Instructions

We are using a genetic algorithm to design interactive animated graphics. Your evaluation of the graphics will guide which ones are deleted forever and which ones stay to populate the next generation. Follow the link to our web site which will display interactive graphics. You can interact by clicking in various places in the graphic.

For each of **FOUR** graphics, click the FINISH button to rate your experience and submit. You will receive an 8-character CONFIRMATION CODE. Please paste it into the box below to receive credit.

You must evaluate 4 different graphics and receive 4 different confirmation codes to receive credit for this HIT. You may have to reload the page to see the next graphic.

Make sure to leave this window open as you complete the survey. You will return to this page to paste each code into a box.

Link:	http://geneticcrowd.appspot.com/eval
Provide confirmation code 1 here:	
Provide confirmation code 2 here:	
Provide confirmation code 3 here:	
Provide confirmation code 4 here:	
Optional: let us know of any difficulties, suggestions, or other comments.	

Figure 1: Amazon Mechanical Turk HIT

IAG is presented to the user through our website, and the user is asked to interact with it and perform a standard one question subjective evaluation. Once the required number of evaluations have been performed for each IAG, the system automatically determines the fitness of each IAG and evolves the new generation. The new generation is evolved from the old generation by either carrying forward survivors from the old generation as they are, by mutating the survivors of the old generation to generate new members, or by creating random new members. The top 20% of the IAGs of the previous generation are carried forward to the next generation unchanged since they represent the best possible solutions to the problem. The next 40% of the new generation are obtained by mutating the top 20% of the previous generation by randomly changing their properties. Finally, the last 40% of the new generation is populated by randomly generating new IAGs. This is done so as not to limit the generated solutions to a small area of the entire search space of possible solutions.

By keeping the top 20% of the previous generation unaltered, we can determine if there is a highly fit IAG that persists across generations. The next 40% allows us to take the highest performers and slightly change their features, and potentially find solutions that have better fitness than the previous solutions. Populating the remaining generation with random new members allows us to explore more of the search space and potentially find a novel solution that would not have been possible through mutation. These are generally accepted as standard mutation practices.

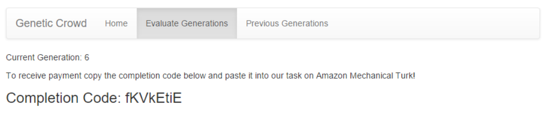
Evaluation

As mentioned in a previous section, we track various amounts of information about each user's interaction with the IAG that is presented to them. The primary evaluation metric used is the evaluation score that is submitted by the participant after he or she has finished interacting with the IAG. This score assists in determining the fitness of the IAG and ultimately whether or not it will make it into the next generation. We require that each IAG be evaluated at least four times, and we take the average of those evaluations to determine the fitness. This is done to increase granularity of fitness and to ensure our solutions are those IAGs which are enjoyed by more than a single individual. The purpose of the experiment is to optimize the subjective appeal of the IAGs. The secondary evaluation metrics such as number, location, and time of clicks are used to determine the relationship of these objective metrics with the subjective ratings. Our findings are described in the results section of this paper

Amazon Mechanical Turk HITs

We integrated our website with Amazon Mechanical Turk and tasked Turkers with evaluating our candidates in each generation. Figure 1 shows the HIT presented to the workers in its final iteration. After each evaluation, a unique confirmation code is generated by the server and presented to the user. This is shown in Figure 2. The worker has to enter this confirmation code into a field on the HIT's page in order to be compensated for the work performed. We created multiple HITs that asked the workers to evaluate several IAGs. To compensate for the volume of evaluations we required per IAG, we asked each Turker to rate four IAGs in a single HIT. Since each generation contained twenty IAGs requiring a minimum of four evaluations each, we needed a minimum of twenty workers to complete our HIT in order to generate the next generation of citizens.

The original HIT contained confusing language and resulted in poor performance on the HITs and many incomplete tasks. In the second iteration of the HIT we strived to


Genetic Crowd Home Evaluate Generations Previous Generations

Current Generation: 6

To receive payment copy the completion code below and paste it into our task on Amazon Mechanical Turk!

Completion Code: fKVkEtiE

Figure 2: Confirmation code given by the server

provide clearer instructions for the workers, and this resulted in an almost 100% HIT completion rate. The second iteration of our HIT also contained a comments box that allowed Turkers to provide us feedback on the task. Some of these comments highlighted the shortcomings of our website. This included the asking by one Turker for ”[a] reload button or icon” to allow continuous evaluation without refreshing the page. Another Turker noted the low pay of the HIT for the perceived amount of work. We found that average completion time for our task was higher than we expected and subsequently raised the compensation to match. Many workers found the task to be ”interesting” and ”cool”.

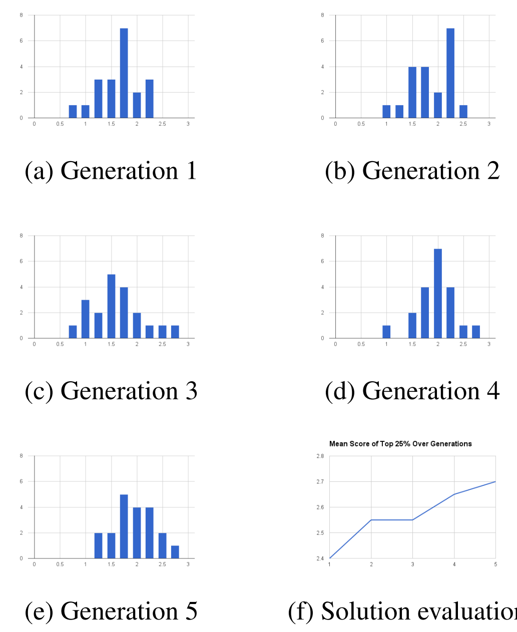
Results

To assess the effectiveness of our combined system, we test the hypothesis that the fitness of the evolved solutions rises over time. This would confirm both that the genetic algorithm is in fact moving its population in the direction of higher average aesthetic assessment and also that the human computation interface we use is effective in providing a useful fitness measure to the genetic algorithm. We also conduct a secondary check on the agreement among workers to confirm that the aesthetic evaluations are not arbitrary. Finally, we examine the objective variables collected and their relationships with the primary subjective fitness measure to assess their potential as inputs to the fitness function.

Evolution

To confirm that evolution improved the fitness of the solutions over generations, we look at the distribution of average subjective evaluations over the members of each population, as shown in figure 3. Because the evolutionary algorithm only keeps its fittest members, which we designate as the solutions of any given population, we can ignore the lowest-scoring majority of the population consisting of newly randomly initialized or badly mutated IAGs. In our analysis, the top 25th percentile of the population by fitness is considered as the solution subpopulation, and it is these solutions in which we hope to see improvement over the generations.

To assess the rate of improvement in the solution subpopulation, we model these assessments as a Hidden Markov Model (HMM) (Zucchini and MacDonald 2009). In each generation t, each observed average evaluation $y_{k,t}$ for an IAG k in the solution subpopulation is treated as a Gaussian random variable which depends on the hidden state μ_t representing the true average, also treated as a Gaussian random variable. This true average is an autoregressive AR(1) process changing from generation to generation by the evolution rate β. Thus, the time series model of evaluations of the solution subpopulation is described by the following equations:



(a) Generation 1 (b) Generation 2

(c) Generation 3 (d) Generation 4

(e) Generation 5 (f) Solution evaluations

Figure 3: a-e: Fitness histograms for each generation; f: Fitness of top 25th percentile over generations

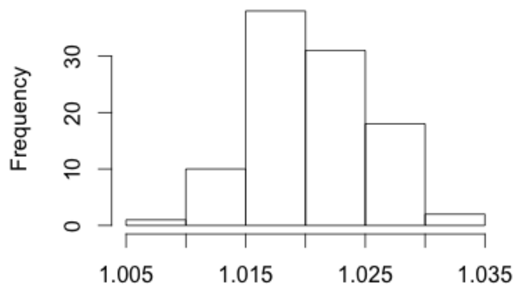


Figure 4: Posterior Distribution for β

$$y_{k,t} \sim N(\mu_t, \sigma)$$
$$\mu_t \sim \beta\mu_{t-1}$$

Using estimates for prior parameters such as the noise parameter σ calculated from the first generation's observations, we perform Markov chain Monte Carlo (MCMC) sampling to estimate the posterior distribution of the evolution rate β (Lunn et al. 2000). The resulting distribution has mean greater than 1 and does not overlap 1 in the 95% credible range, as shown in figure 4. This confirms that the evolution rate of the solutions in our model is greater than one and that this result is not coincidental.

Agreement

As further confirmation that the aesthetic evaluations made by workers are not arbitrary, we investigate the degree of agreement between workers. Because subjective evaluation

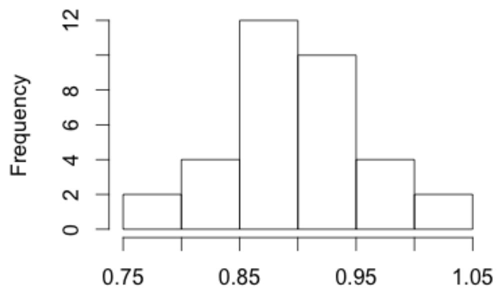


Figure 5: Distribution of Random Disagreement Scores

tasks lack a single gold standard, we hypothesize evaluations are distributed according to *schools of thought*. This is when distinct groups of workers emerge, with agreement within each group but disagreement between groups. In (Tian and Zhu 2012), schools of thought are modeled as a Dirichlet Process, and posterior estimates of the parameters are used to evaluate worker dependability and task clarity. We take a simpler approach in which we propose to score the total disagreement D in a population as follows:

$$d = \sum_{i}^{N} \sum_{j>k} |v_i j - v_i k|$$

Here $v_i j$ represents the evaluation of worker j for artifact i. Thus we are simply taking the total sum of the absolute differences in evaluations for distinct pairs of workers across the population of artifacts. To evaluate this disagreement score we run several Monte Carlo simulations in which simulated workers choose random evaluations for each artifact, then collect the distribution of random disagreement scores D_r. We would expect d to fall on the far low end or outside the credible range of D_r.

The disagreement score we calculated for our final generation was 0.85. Although this is, as expected, on the low end of the distribution of random disagreement scores as seen in figure 5, it is not outside the 95% interval, suggesting it would not be infeasible that many of our workers' evaluations were made arbitrarily.

One caveat to our approach to estimating task clarity according to worker disagreement is that the number of evaluations made by each worker is only four, whereas there are a total of twenty artifacts per evaluation. In (Tian and Zhu 2012), each worker is expected to complete all tasks. It may be the case that this deficiency is costing us statistical significance in our disagreement results. Nevertheless, together with the significantly positive estimate for the evolution rate explained in the previous section, the fact that our final generation's disagreement score is in fact on the low end of the random distribution supports our confidence that the worker's aesthetic evaluations are mostly non-arbitrary.

Click Data

One of our research questions was about using objective data collected from our workers to inform evolution of the subjective measure. In addition to collecting subjective evaluations, we also collected the workers' click data. While this was not used as input to the fitness function in our experiments, we examine the distribution of this data over the generations and compare it to that of the subjective evaluation data. This provides some insights into how this objective data may be used in the future to help evolve subjectively evaluated artifacts.

First we transform our click data into two measures for each evaluated IAG: average number of clicks c and average time spent in observation t. We assume c to be a measure of interactivity and t to be a measure of engagement.

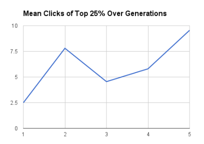
Our first finding is that both c and t are negatively correlated with evaluation scores in the early generations and positively correlated with evaluation scores in only the final generation. This is a surprising result as we were expecting these measures to correlate positively, if only roughly, with the subjective measure throughout the entire process. One possible explanation is that these measures could correspond to a worker's frustration with an artifact just as easily as they could correspond to pleasure. Regardless of the explanation, it seems that these objective measures cannot be naively used as surrogate fitness measures when trying to optimize subjective evaluation through evolution.

It may be possible to process and refine these measure to make them useful. Interestingly, when looking at only the top 25th percentile of any generation, these measures rise significantly over the generations just as the subjective evaluation does, as shown in figure 6. We hope that - perhaps in combination with other features and with some additional processing to address outliers - these objective measures could be used to find an indicator correlated well enough with the subjective evaluations to be used in their stead. We leave this question for future work.

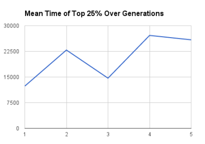
Future Work

In terms of future directions of work we have different aspects of the system we can improve upon and experiment with. Currently, we generate very elementary artifacts that are evaluated by the human workers. In subsequent iterations of the system we would like to incorporate the genetic algorithms developed by Togelius and Schmidhuber (Togelius and Schmidhuber 2008) to generate games based on an objective fitness function and complement it with a subjective one using the system we developed to gather evaluations from the crowd.

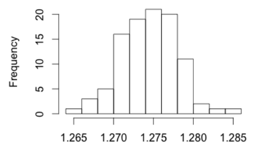
The design of the HITs posted on AMT also could incorporate more information describing our research goals so as to provide better context to the worker and increase engagement. A number of workers perform the tasks sub-optimally if they do not know the context or reasons as to why the task has been posted and the goal it is trying to accomplish. We also need to implement better mechanisms to check that the work being performed by the worker is being done so correctly and not in a hurry or by a bot. This is a difficult



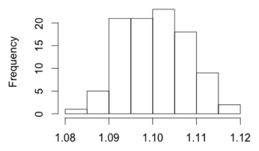
(a) Solution clicks



(b) Solution times



(c) Posterior for β_c



(d) Posterior for β_t

Figure 6: a-e: Fitness histograms for each generation; f: Fitness of top 25th percentile over generations

but important task as we rely on truthful subjective feedback from the user after he or she has interacted with the object for a decent amount of time. If the feedback is not truthful or is submitted by an automated bot then the data being collected would be tainted.

In the future we also aim to incorporate the other data collected regarding the evaluation of a candidate by the human worker into the fitness function. While currently our fitness function is based on a single piece of data, we would like to extend it to take into consideration various other attributes of data collected from the user's evaluation. It would be interesting to see whether we can make effective use of the interaction data collected resulting in a better set of candidates during mutation.

The Genetic Crowd system developed by us could also be used in various other domains that use genetic algorithms to generate potential solutions, especially areas or problems in which the solutions cannot easily be evaluated using an objective fitness function. The easy availability and low-cost of the crowdworkers on AMT enables researchers to rapidly collect subjective evaluations in order to evaluate the fitness of an object.

Conclusion

We have presented a system which enables the use of subjective fitness functions in genetic algorithms using data gathered from the crowd using the AMT platform. The Genetic Crowd system has shown that it is possible to use a crowd of human workers to subjectively evaluate candidates of a genetic algorithm and guide the evolution of subsequent generations. The results demonstrate the fact that subjective fitness functions can be developed and can be evaluated using crowd workers. The system developed by us can be generalized to be used by various other domains of problems that use genetic algorithms and do not have a good objective fitness function.

In the future, we would like to improve the genetic algorithm in order to generate more interesting artifacts such as Atari type games or animations. If successful in the future we could develop a game automatically based on genetic algorithms and use the crowd to guide the evolution process thereby resulting in games that are developed using crowdsourced workers.

Genetic Crowd provides a practical way of evaluating candidates in a genetic algorithm using subjective fitness functions. We have shown that it is an effective strategy for guiding the evolution of genetic algorithms.

References

Alonso, O. 2009. Guidelines for designing crowdsourcing-based rlevance experiments. *Magnetic Resonance in Chemistry* 1–2.

Ciurana, E. 2009. Google app engine. *Developing with Google App Engine* 1–10.

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

Lunn, D. J.; Thomas, A.; Best, N.; and Spiegelhalter, D. 2000. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing* 10(4):325–337.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65(6):386.

Sakamoto, Y.; Nickerson, J. V.; and Bao, J. 2011. Evaluating Design Solutions Using Crowds. In *AMCIS 2011 Proceedings*, 1–9.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8. IEEE.

Secretan, J.; Beato, N.; D'Ambrosio, D. B.; Rodriguez, A.; Campbell, A.; and Stanley, K. O. 2008. Picbreeder: Evolving Pictures Collaboratively Online. In *Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08*, number Chi, 1759–1768. ACM Press.

Tian, Y., and Zhu, J. 2012. Learning from crowds in the presence of schools of thought. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 226–234. ACM.

Togelius, J., and Schmidhuber, J. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*, 111–118. Ieee.

Zucchini, W., and MacDonald, I. L. 2009. *Hidden Markov models for time series: an introduction using R*. CRC Press.